An Evolutionary Approach to Automatic Video Editing

Andrew Mansfield

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2009
This dissertation 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:
An Evolutionary Approach to Automatic Video Editing

Submitted by: Andrew Mansfield

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop). This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

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:
Abstract

The casual capture of consumer video is increasingly common in modern life. We present an automatic editing system which attempts to bridge the gap between ‘interest sparse’ raw footage and tightly edited, salience rich clips. We evaluate Genetic Programming as a novel method of video editing and present an entirely new representation of the editing process as operators within evolving programs.

We evaluate the system against a set of sample videos and find that the core system of cutting footage works well but that attempts to simulate camera techniques such as panning and cropping are unsuccessful.
Contents

1 Introduction 1
  1.1 Problem Description .............................................. 1
  1.2 Contributions .................................................. 2
  1.3 Applications .................................................... 2
  1.4 Dissertation Structure ........................................... 3
    1.4.1 Literature survey ........................................... 3
    1.4.2 Project planning ............................................ 3
    1.4.3 Core .......................................................... 3
    1.4.4 Evaluation .................................................. 3
    1.4.5 Conclusion ................................................... 3

2 Literature Survey 4
  2.1 Introduction ..................................................... 4
  2.2 Film Editing ..................................................... 4
    2.2.1 Definitions .................................................. 5
    2.2.2 Film Grammar ................................................. 5
  2.3 Automatic Editing Systems ........................................ 6
  2.4 Computer Vision Techniques ..................................... 7
  2.5 Genetic Programming ............................................. 8
    2.5.1 Justification ................................................ 9
    2.5.2 The Process ................................................. 10
    2.5.3 Video Editing through Genetic Programming ............... 10
    2.5.4 Implementing a genetic programming system .............. 11
2.6 Conclusion .................................................................................. 13

3 Design Process ............................................................................. 14
  3.1 Software Process ....................................................................... 14
    3.1.1 Core Deliverable .................................................................. 14
    3.1.2 Extensions ......................................................................... 15
  3.2 Requirements ........................................................................... 15
    3.2.1 The automatic video editing system .................................. 15
    3.2.2 The visual analysis module ............................................. 15
    3.2.3 The Genetic Programming Framework ......................... 16
    3.2.4 Non functional requirements ......................................... 16
  3.3 Resources ............................................................................... 17
    3.3.1 Programming Language .................................................. 17
    3.3.2 Software .......................................................................... 17
  3.4 Overall Design .......................................................................... 18

4 Automatic editing system ............................................................. 19
  4.1 Genetic Programming Framework ......................................... 19
    4.1.1 Genetic Program Representation ...................................... 19
    4.1.2 Population Initialisation .................................................. 21
    4.1.3 Breeding .......................................................................... 21
  4.2 Genetic Programming Representation ................................... 22
    4.2.1 Editing as a program ....................................................... 22
    4.2.2 Constant nodes ................................................................ 25
  4.3 Video sequences ...................................................................... 30
  4.4 Cropping ................................................................................... 33
  4.5 Panning ..................................................................................... 36
  4.6 Conclusion .............................................................................. 40

5 Evaluation ................................................................................... 41
  5.1 Split ......................................................................................... 41
    5.1.1 Video 1 ............................................................................ 41
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The gradings of shot distance</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Upper body detection</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>The crossover process</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>The mutation process</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>The core design</td>
<td>18</td>
</tr>
<tr>
<td>4.1</td>
<td>The software structure</td>
<td>20</td>
</tr>
<tr>
<td>4.2</td>
<td>An individual edit process (simple input output diagram)</td>
<td>23</td>
</tr>
<tr>
<td>4.3</td>
<td>The split operator</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>A resultant sequence for split</td>
<td>24</td>
</tr>
<tr>
<td>4.5</td>
<td>Problematic selected clip.</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>Split node with constant node</td>
<td>26</td>
</tr>
<tr>
<td>4.7</td>
<td>Output from split with constant</td>
<td>27</td>
</tr>
<tr>
<td>4.8</td>
<td>Maximum fitness</td>
<td>28</td>
</tr>
<tr>
<td>4.9</td>
<td>Fitness std. deviation</td>
<td>28</td>
</tr>
<tr>
<td>4.10</td>
<td>The two problematic local maxima</td>
<td>30</td>
</tr>
<tr>
<td>4.11</td>
<td>Upper body detection</td>
<td>31</td>
</tr>
<tr>
<td>4.12</td>
<td>A representation of the initial edit program</td>
<td>32</td>
</tr>
<tr>
<td>4.13</td>
<td>Results for split against sample video</td>
<td>33</td>
</tr>
<tr>
<td>4.14</td>
<td>Explanation of crop</td>
<td>35</td>
</tr>
<tr>
<td>4.15</td>
<td>Example of results for crop</td>
<td>36</td>
</tr>
<tr>
<td>4.16</td>
<td>Example of results for crop</td>
<td>36</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

4.17 Explanation of panning .......................... 37
4.18 Non terminal pan representation ................ 39
4.19 Testing pan representations ...................... 40

5.1 Video 1 output ........................................ 42
5.2 Fitness value ......................................... 42
5.3 Standard deviation .................................. 43
5.4 Video 2 output ........................................ 44
5.5 Fitness value ......................................... 45
5.6 Standard deviation .................................. 45
5.7 Video 3 output ........................................ 46
5.8 Fitness value ......................................... 47
5.9 Standard deviation .................................. 47
5.10 Video 4 output ....................................... 48
5.11 Fitness value ......................................... 48
5.12 Standard deviation .................................. 49
5.13 Video 5 output ....................................... 49
5.14 Fitness value ......................................... 50
5.15 Standard deviation .................................. 50
5.16 Video 5 output with adjusted fitness constants .. 51
5.17 An output frame with accompanying detection (blue) and cropped frame (red). The portion in red indicates the chosen portion of the screen. .... 52
5.18 An output sequence for the non-terminal representation of the pan operator .... 52
5.19 Maximum fitness for the non-terminal pan representation ............... 53

A.1 Gannt chart for the project ......................... 60
List of Tables
Acknowledgements

I would like to thank my partner Nicky, without whom none of this would have happened. I would also like to thank John Collomosse for all his help and support. Big thanks also to my daughter Millie for making such a good subject for filming!
Chapter 1

Introduction

Following in the footsteps of digital photography, which is now almost ubiquitous in the public domain, affordable digital video is becoming increasingly accessible to the consumer. The advent of video-capable mobile devices has led to an explosion in the amount of video captured. The large amount of user generated video created is evident in video sharing sites such as YouTube.

1.1 Problem Description

As is the case for digital photography, the capture of digital video is mostly casual in nature. In the general situation a user attempts to record a real life situation on film. They are in a passive position in that they do not know how events will unfold and so must record large amounts. This will often lead to periods of uninteresting footage interspersed with the periods of interest it was the user’s intention to capture. The video may also have other problems related to the casual nature of the capture, such as poorly framed subjects and shaky camera work. These problems can lead to videos which are not enjoyable to watch, despite the periods of interest within them. Unlike digital photos which have innate accessibility due to their immediacy, user-captured digital video is much less frequently watched in its raw form.

In order to be accessible the raw captured footage needs to be edited so as to remove the uninteresting and low quality footage and leave a resultant sequence containing only the periods of highest interest. One possible solution is to manually edit the footage and there are many applications capable of video editing. Products aimed at the home market such as the Apple iMovie aim to simplify the process as much as possible, but manual editing is at its heart a tedious and time-consuming process. This is a significant barrier to the undertaking of a manual edit. It must also be considered that even if the will is there, the outlay of time required for a manual edit may simply not be appropriate for a piece of...
casual footage.

In order to make casual editing a much more viable activity an automatic system is required which can take a set of raw footage and automatically assess the footage and generate an optimised edited version which is of highest interest value to the user. Research in this area dates back to the late 90s (DeMenthon, Kobla and Doermann, 1998) and indeed a commercial offering exists, Muvee®. This system does automatically edit footage although its shot selection is based on a series of simple stylistic rules. Indeed Muvees falls back on manual editing to allow the user to select important sections of their raw footage. Indeed very little of the published research has considered applying artificial intelligence techniques to the problem, tending instead to use rule based systems or mathematical calculations to decide the finished edit.

1.2 Contributions

The process of editing video footage can be thought of a series of operations (be it a cut or zoom) on an unedited sequence. It can be considered in some ways to be like a computer program. The novel hypothesis of this project is that these edit programs could be evolved into programs that create edits of high quality unedited footage. The process of evolving programs is called genetic programming.

The overall aim of this project is to build and test the viability of an automatic video editing system using a genetic programming system to make the decisions about footage selection. Genetic programming is a system of artificial intelligence that draws on the theory of evolution as its inspiration. Programs to complete a task are bred and then by a selection process based on a ‘fitness’ criteria the programs are evolved until a sufficiently successful program has been created. The application of genetic programming to video editing is a novel concept and is not in the literature.

In addition the means of measuring the fitness of a video sequence are explored, such that this ‘fitness’ measurement can be used to evaluate the programs generated within the genetic programming framework. This idea involves evaluating what are the interesting elements of a video sequence and whether we can match these elements to techniques from the field of computer vision. It must be recognized that this idea of interest is highly subjective and that any number of solutions could be equally valid.

1.3 Applications

The on-line video sharing site YouTube contains a vast amount of user generated video footage, a large proportion of which is in an unedited raw form. A hugely useful application would be the integration of the system within YouTube. An uploading user could upload

---

2Muvee autoProducer 6: www.muvee.com
3How successful the program is in completing the task
a video to the site and then request that the video be automatically edited. Alternatively a viewer could watch an edited version of unedited content stored on the site. Integration with a manual video editing application such as Apple’s iMovies could create a much simpler and less time consuming editing process. Various degrees of semi-automation could be used. As an example, the system could be made to generate a selection of individual candidate clips of high interest levels and then present them to the user. The user could then use a selection of candidate clips in their video.

1.4 Dissertation Structure

1.4.1 Literature survey

We discuss genetic programming; its workings and its suitability for the task in hand. We investigate the idea of elements of interest within video footage and the means of measuring them.

1.4.2 Project planning

We give an explanation of the ‘exploratory iterative’ development procedure, an outline of a high level set of requirements and a design overview.

1.4.3 Core

We discuss the development of the genetic programming system; the representation, the breeding strategies and the fitness function.

1.4.4 Evaluation

The system is tested against a series of real videos and data sets and the results are analyzed.

1.4.5 Conclusion

We critique the process and outcomes of the project and consider ideas for future work.
Chapter 2

Literature Survey

2.1 Introduction

In this survey we attempt to outline a few of the interesting aspects of film editing discovered and how they are applicable to the project. We also give a short discussion of existing automatic film editing systems and computer vision techniques. We give an account of Genetic Programming and its intended use.

2.2 Film Editing

The editing of recorded film for the improvement of dramatic effect has been around since the early 20th century with films such as D.W Griffith’s “Birth of a Nation” already utilising many of the editing techniques still employed today. Indeed a text on the subject from 1935 (Spottiswoode, 1935) is still largely valid in the current age.

The field of film editing is a component of an art form, all be it a very technical component. It is therefore subjective but there are still some common aims:

- The clarification of the delivery of a narrative structure.
- To optimize the quality of the composition.
- To maintain the interest of the audience.
- To remain as unobtrusive as possible.

Given that all the analysis of the video within the project will be conducted by computer vision algorithms we assert that the current semantic understanding of video is too primitive to be able to form any understanding of the story within a video. We must therefore discount any idea of manipulating narrative and stick to the more technical elements of the editing, primarily that of interest.
2.2.1 Definitions

There is a certain amount of basic terminology which is extremely useful for this project:

**Shot**  An individual element of continuous film footage.

**Scene**  A collection of shots edited together to form a coherent whole.

**Master shot**  An overarching shot which is used to establish the make-up of the scene.

**Cut**  An instantaneous switch from 1 shot to another.

**Pan**  A horizontal camera movement.

**Shot Distance**  This defines 5 gradings of shot distance which are context sensitive in the sense that a close shot of a house will differ from a close shot of a person. I give a description of each for the human form.

- **Close up**  A standard head shot.
- **Close shot**  A shot cut off under the shoulder.
- **Medium shot**  A shot cut off under the waist.
- **Full shot**  A full body shot.
- **Long shot**  A full body shot occupying much less of the screen.

![Figure 2.1: The gradings of shot distance](image)

2.2.2 Film Grammar

(Arijon, 1991) and (Spottiswoode, 1935) discuss the idea of a “grammar of film”, a set of structures that can be used to capture a particular situation in a way that will be most effective at pleasing and keeping the interest of the audience.

The majority of the ideas detailed within (Arijon, 1991), such as the established ‘Triangle Principle’, require two or more camera positions. Given that we are dealing with the material recorded from a single viewpoint, none of these are possible. However there are elements of the grammar which are directly applicable. As long as the shot structures keep to the same visual axes then we can cut from within the frame of the raw video to create closer shots.
An element of this is that in the field it has been discovered that when capturing the human
form on camera there are five standard ‘cutting heights’ which will be more aesthetically
pleasing on screen: under the armpit, under the chest, under the waist, under the crotch
and under the knees.

Another is that a static master shot of a scene can act as an ‘establishing shot’ which will
allow an audience to become accustomed to a scene. This shot however if left too long will
become stagnant and boring and so should be broken up using closer shots. In fact cutting
should be used in this way with all shots.

It is important to make sure that cuts between shots are matched. There are three criteria
to this matching:

**Position** The position of the subjects of interest should be at least in some sense matched
on the boundary between shots to avoid too much visually jumping in the screen
which is abrupt and confusing to the audience.

**Movement** The movement of subjects or the camera should be matched between shots,
or at least not at complete odds as this is again abrupt and draws attention to the
camera.

**Look** Deals with the direction of gazes, which cannot really be used within this project.

With all of these ideas one must realise that this project is at a tangent to the conventional
filmmaker. When using prerecorded casual amateur footage we lose the ability to capture
footage from multiple angles (as already discussed), we have no control over the motion
of the camera and we have no overlap of the chronology so no repetition of events from
different positions.

This adds a lot of difficulty to the task of creating a video of high interest value. However
we may be able to apply some of the simpler ideas such as shot matching.

### 2.3 Automatic Editing Systems

The theme of this project has been an area of active research since the early 1990’s. The
various avenues explored include many areas such as:

**Video summarization** This is the process of finding key aspects of a video mostly man-
ifested as key frames of the video. (DeMenthon et al., 1998) identifies important key
frames in a video by pulling out important information from the video stream and
modelling it as a trajectory curve in a multidimensional space.

**Automated Film Direction** There are variations within this field including systems
such as (Hospedales and Williams, 2008) which attempts to use Bayesian learning
to direct films while being instructed by intermittent expert human intervention.
Video editing assistance The system within (Girgensohn, Boreczky, Chiu, Doherty, Foote, Golovechinsky, Uchihashi and Wilcox, 2000) works by assessing the suitability of each individual frame using a variety of low level measures. It then picks out shot boundaries based on the edges of regions of high suitability. The system also allows the shot boundaries to be adjusted by the user using a spring based algorithm.

Automatic Video Editing A recent example of automatic video editing is (Hua, Lu and Zhang, 2005) which is close to what we attempt to achieve. It calculates shot boundaries automatically by using low level analysis of the frames looking for positions where shots can end naturally. It then views the shot selection as an optimization problem and attempts to solve it using a genetic algorithm. Major points to note is that one of the measurements of the film is sentence detection as the researchers found that one of the most unpleasant features for the audience is split sentences. It also measures the quality of clips based on the idea of an attention model which combines the features thought most important by the researchers.

2.4 Computer Vision Techniques

One of the most important decisions to make for this project is which tools to use in order to interpret the video footage. In order to decide this we must settle on what is important for a good video. The first criteria will be elements of interest within a frame. We conclude that people are the most interesting elements within amateur video. Detecting people can be achieved in a number of ways, the most common of which is face detection. Advances in face detection have meant that real time systems of face detection are possible, such as(Viola and Jones, 2004). These systems use classifier based systems to identify important features within frames and use these for detection. In (Viola and Jones, 2004) a cascade structure is used to quickly discard any features detected which are not in critical areas of the image. This allows the system to detect faces extremely quickly which is a large boost for video processing.

The problem with these detection systems is that they tend to base there detection on a fairly rigid model of the face and so can only really identify near camera facing faces. This may not be such a major problem as a detected face will mean that the subject is well presented. This means that a shot with a detected face will be of high interest value.

More recent research into upper-body detection (Ferrari, Marin-Jimenez and Zisserman, 2008) will also be a useful measure of interest. The algorithm is able to track torsos from the front or behind with a high degree of success. Given that user created video is likely to frame persons in a wide variety of poses we are most likely to get a lot more detection within videos with this. It will most likely form the central detection element of our system.
CHAPTER 2. LITERATURE SURVEY

These two elements should work in tandem to allow us to have some reference of where persons are on screen. Given information relating to the persons position and size we should be able to estimate their position on screen and this may allow us to discern whether they are within range of one of the preferred ‘cutting heights’ which would be of higher value than if not.

More general measures of the video will be required where person detection fails or indeed there are no persons to detect. Even when persons are present they will still be a useful measure.

Optical Flow (Nikos Paragios, 2005) is a measure of the movement of ‘features’ within a video and we can identify movement as interesting within our system and use it to identify interesting areas of a scene using the optical flow of the features in that region as a measure.

In the field of scene segmentation (Yin, Criminisi, Winn and Essa, 2007) introduces the interesting concept of motons. These are classifications of pixels by means of there movement and whether they are close to any edges within the image. This can lead to some interesting grouping of different motons within a scene. If we could identify the most interesting type of moton within a scene we could measure the interest value of a region of the screen by the density of that type of moton.

2.5 Genetic Programming

Once we have identified the features of interest from within the video we must then set about performing the editing on the video. This is an extremely difficult problem as there is no correct answer to what is the absolute best video from a set of video footage. The
The method suggested for the problem, and one which seems extremely well suited to it, is genetic programming. Genetic Programming, which began with (Koza, 1990), is a very successful offshoot of the field of evolutionary algorithms, within which a population of potential solutions are evolved in order to improve the quality of the solutions. The main difference between evolutionary algorithms and genetic programming is, quoting (Poli, Langdon and McPhee, 2008):

In genetic programming we evolve a population of computer programs.

The programs themselves are the members of the population. To create programs that can be easily evolved most systems implement them as trees, similar to the abstract representation used in common parse trees, which contain operators and constants as their elements.

2.5.1 Justification

Genetic programming is the search technique we have chosen to investigate and there are good reasons to suggest it will be a viable technique for the problem at hand. (Poli, Langdon and McPhee, 2008) gives a set of criteria identifying problem types for which genetic programming could be successful. Among these are:

*The problem domain is not well understood.* The problem domain of video editing is certainly not well understood and there have only been a limited number of attempts at analytical solutions.

*Finding the size and shape of the solution is a major part of the problem*. The search space of possible video edits is large and the edits we create will certainly vary in size and shape a great deal. Indeed finding the size and shape of the edit is the problem to be solved.

*Significant amounts of test data exist in computer readable form* With video as our source material there will be the opportunity to generate large amounts of test data based on computer-vision based measurements.

*Testing the quality of solutions is easy yet this does not translate directly into methods to obtain such solutions* It should be relatively simple to generate analytic measures of the quality of created video edits. Transferring these to the creation of video edits is hard and will require the use of genetic programming.

*Approximate solutions are acceptable (or the only likely result)* The solutions we generate will be judged on largely aesthetic measures, therefore there is no distinct and definite ‘best’ solution to the problem at hand.
2.5.2 The Process

A typical scenario for utilising genetic programming is as follows:

1. The problem is described by way of a fitness criteria which will be used to judge the results obtained from the individuals within the program population. These criteria define the high level goal of the system. We may also require a set of data in order to apply the fitness criteria (in the form of a ‘fitness function’).

2. A set of operators and constants are defined. These will form the basic building blocks from which the programs will be built.

3. A strategy for selecting the fittest members of the population is determined by way of their results when applied to the fitness criteria.

4. A breeding strategy is created whereby the program population is evolved by means of combining and altering the surviving fittest members of the population.

5. An initial random population of programmes is created.

6. A loop is run until a certain fitness threshold is reached:
   - All members of the programme population are run.
   - The results of the run are measured using the fitness function.
   - A sample of individuals are selected from the fittest members of the population.
   - The selected members are bred using the breeding strategy to form a new population.

7. The results of members which have achieved the fitness threshold are returned as solutions of the problem.

2.5.3 Video Editing through Genetic Programming

To utilize such a system we must consider the elements with respect to my own problem:

- The fitness function of the system will have to be some linear combination of the detected faces, torsos and objects, along with some more general measures of interest such as optical flow or possibly some measure of the more interesting motions. This will have to be tempered by penalties for overly long or short cuts or possibly too much

- The operators will include a set of operators which will perform the basic operation of cutting footage. Other possible operators will include pan and zoom operators, although reading from (Arijon, 1991) suggest these will be of much smaller importance than the simple cut operator.
2.5.4 Implementing a genetic programming system

There are many different factors which must be considered when implementing genetic programming. Firstly one must decide upon a strategy for creating the initial population. There are two main strategies for this: ‘Full’ and ‘Grow’, both creating trees of a maximum depth.

With ‘Full’ the tree is created and random from non terminal operators, until the maximum depth is reached, at which point the remaining tree is completed using terminal members of the operator set. This creates trees with leaves all at the same depth.

With ‘Grow’ both non terminals and terminals are used up until the depth limit, at which point the tree is completed using terminals. This creates trees which can be asymmetrical in structure.

Both these methods produce very different types of trees, and so a common method of initial population creation, suggested in (Koza, 1992), is named ‘ramped half and half’. As the name suggests this involves the use of both methods, the ‘ramped’ refers to the fact that trees that are generated for a range of maximum depths. This allows for a good amount of variety among the initial population.

Then there are the breeding strategies to consider. The most common approach is to utilise a combination of two techniques: crossover and mutation.

Crossover is the process of taking two programs from those selected from the previous generation and switching random parts of their program trees to create new trees for the new generation. This makes up the majority of the breeding process. Due to the fact that the switch is largely random, all operators within the framework must accept and return variables of the same type.
CHAPTER 2. LITERATURE SURVEY

Figure 2.3: The crossover process

Mutation makes up a much smaller part of the breeding process. Again we take members selected from the previous generation and change part of their program tree. This could take the form of altering constant values in the chosen part of the tree or even creating a entirely new random sub-tree to replace the chosen part of the tree. Mutation is seen as a much more random element than crossover (Crossover is taking parts of two trees selected via fitness, whereas the introduced material in mutation is random and has no specific value to add) but it is still used in moderation to add variation to the population.

Figure 2.4: The mutation process

Another possibility is elitism whereby the fittest members of the previous generation are added to the new generation directly. This must be used sparingly as otherwise elite members and their offspring can quickly swamp a population leading to a lack of variation.
This can impair evolution of the population.

One of the main problems affecting genetic programming is bloat, which is the progressive growth in size of the members of the program population as the generations pass. This can lead to programs with increasing computational expense and make the evolution of the populations extremely slow. The most common methods of limiting bloat are by explicitly limiting the size of the individual members of the population. The only drawback to this is to limit the range of possibilities available within the program population. It has been show however that elitism can be a significant help in reducing bloat (Poli, McPhee and Vanneschi, 2008).

The operator set must fulfil three criteria identified in (Poli, Langdon and McPhee, 2008). Firstly the operators should be type consistent i.e. return values of the same type so as they can be freely interchangeable in breeding. The second point is evaluation safety which means any operator should return a value on any input. Finally the operator set should be sufficient; it should have enough expressive power to generate any possible solution to the problem.

2.6 Conclusion

We have given justification to the idea that Genetic programming is ideally suited to the problem at hand. The search space of possible videos is so large and also because the problem of video editing fits most of the criteria for which genetic programming would be suited. Automatic video editing is a well researched problem but it appears that we have identified a novel approach. The fitness factor of the system will attempt to tie in the information we extract from the videos with the editing rules that are easily applicable to amateur video footage. This will include upper-body-detection.

One final thing to mention that speech detection would be an extremely useful addition to the detection armoury but must be discounted to scope the project correctly.
Chapter 3

Design Process

This chapter is a discussion the nature of the project and how this informs on development. A high level set of requirements is given and the resources required for the project are documented. The chapter ends with an overall design of the system.

3.1 Software Process

The project undertaken is research rather than development orientated. It is largely exploring new territory, therefore planning and design are outlined at a very high level. The approach to development is an ‘explorative iterative’ approach. Working in cycles ideas are developed, implemented and evaluated. The design of new iterations is informed by earlier iterations. We chose the iterative approach with an achievable set of core objectives, in order to mitigate against the risk of failure. A gannt chart detailing the project schedule is in Appendix A.

3.1.1 Core Deliverable

1. The creation of a video analysis algorithm which can take a given unedited video and return a set of data which represents the features of interest contained within it.

2. The creation of a genetic programming framework containing:
   - Operators that query the data created by the video analysis.
   - Operators that can manipulate the original video in order to edit it.
   - Suitable crossover and mutation strategies in order to evolve the program population.
   - The creation of an ‘interest’ measure which, applied to videos can be used as a measure of fitness for the created programs.
For the core deliverable a few simple measures of ‘interest’ are taken, for the analysis phase and to create the fitness function. The most important measure will most likely be upper body detection. The operators within the genetic programming framework will be limited to those querying the results of the analysis phase and a simple cut operator to generate the finished result.

3.1.2 Extensions

If the core deliverable is completed. Possible extensions include:

1. The investigation of additional measures of ‘interest’ and their incorporation into the editing system. This will in turn lead to improvement in the fitness function used in the genetic programming framework.

2. The creation and evaluation of new video manipulation operators to be used within the genetic programming framework, such as pan and crop.

3.2 Requirements

3.2.1 The automatic video editing system

- The system should take a video of arbitrary length and return an ‘edited’ video.
- The returned video should have an increased, or at worst equal, ‘interest’ density compared to the unedited input video.
- The system should be composed of three independent modules:
  - A visual analysis module which takes an unedited video and returns a set of data (labeled VA) which represents the results of the analysis.
  - A genetic programming framework module which takes the set of data VA and returns a data structure (EV) which represents the structure of the resultant edited video.
  - A video construction module which takes the original unedited video and the data structure EV and returns a completed edited video.

3.2.2 The visual analysis module

- The visual analysis module should be extensible to allow the addition of new measures to be applied to the video.
3.2.3 The Genetic Programming Framework

- The framework should be able to create an initial population of programs to begin the evolutionary process.
- The GP operator set should have the property of ‘sufficiency’, specifically that the representation of a video editing program should allow the creation of successful programs.
- Operators should be *type unified* and *evaluation safe*.
- Any program created by the genetic framework should always terminate.
- Any program created by the genetic framework should take a set of data VA and return a data structure EV.
- The framework should have a suitable ‘fitness’ function which takes a data structure EV and a set of data VA and returns a valuation which represents the ‘interest’ density of the edited video.
- The framework should be capable of selecting a proportion of the program population based on their ‘fitness’.
- The framework should have suitable crossover and mutation functions which can evolve a program population.
- The framework should have an exit condition which causes it to return when either a sufficiently successful program is created or the population has reached a plateau in terms of the success of its programs.
- The framework should always return.
- The genetic programming framework should be extensible to allow the addition of new operators.

3.2.4 Non functional requirements

- There are no time constraints on the execution of the program.
- To aid the iterative development, the genetic programming framework should of generic design in order to allow new operators and fitness measures to be added as development progresses.
3.3 Resources

3.3.1 Programming Language

The programming language used will be C due to its efficiency and simplicity. It has slight drawbacks in that it is a low level language and therefore requires some care when coding.

3.3.2 Software

- **OpenCV:** An open-source computer vision library. Contains functions for manipulating video data which will be necessary for the project.

- **Upper Body Detector:** Software for identifying and tracking upper bodies created by the Visual Geometry group at Oxford University.

- **Visual Studio 2008:** Development environment.
3.4 Overall Design

As described in the requirements the system will contain three distinct sections: the video processing module, the genetic programming framework and the output video writer.
Chapter 4

Automatic editing system

We outline the development of our system and perform some intermediate analysis of our developing system. The system is developed in the C programming language.

4.1 Genetic Programming Framework

We began development with the construction of an implementation of the GP framework. The iterative nature of the development made it essential that modifications and additions to the system should be as simple as possible. We therefore required the system to be as generic as possible. The core elements of the system (Program representation, population initialisation and evolution) had to be generic and independent of the (constantly changing) edit operations and fitness function in order that they could remain constant throughout development.

4.1.1 Genetic Program Representation

Listing 4.1: Node structure

```c
typedef struct node *NODE;
struct node {
    NodeType type;
    NODE kids[4];
    int kid_count;
};
```

The GP programs were designed as simple node-based trees. The node structure contained: an enumerated NodeType value representing the genetic operator type, an array of pointers to child nodes and an integer 'kid_count' so the number of children was held explicitly. This representation is independent of the operator set and any changes to it.
An operator can now be defined by a few simple steps: writing the operator’s underlying functions to the signature of the function pointer, adding a pointer to this function to an array of all the operators and then defining properties of the operator (such as the number of children) in a data structure. To add new operators at a later date requires no modification of the existing operator set. The signature of the function pointer specifies the unified types used in the specified operator set. To execute a node in the tree we simply use the genetic operator type as an index into the array of functions.

Listing 4.2: Function pointer signature

```
<RETURN TYPE> (*funct)(NODE currentOperator, <Parameter1>...);
```
4.1.2 Population Initialisation

To initialize the population we implemented a program tree creation function within the `init_tree.c` module. This was a simple implementation of ramped half and half (Koza, 1992). To keep the initialisation module generic and independent of any changes to the operator set we made the initialisation function ‘ask’ the operator module for nodes through a set of unchanging functions in the fixed ‘ops.h’ header. Thus if a non-terminal node was required it would call the `make_NonTerminal` function which would return a random non-terminal node along with the number of children it required. The initialisation function would then fill in the children in a recursive manner, not needing any information about the operator set.

4.1.3 Breeding

The core of the system is the evolve module. The evolve module stores the population arrays of GP programs; there are two arrays so that each new generation can be bred from the old generation in a separate array. One array is populated by the `init_tree` module. The array is then passed to the fixed rank module which uses the current fitness function module (through the fixed fitness.h header) to rank the population in order. The results are stored in the evolve module in a separate ranking-array which holds simple ‘rank’ structures containing a pointer to a GP program tree and the fitness score for that tree. The ranking-array is sorted on the scores. Parent selection for breeding is then handled stochastically on the ranking-array, using a randomly generated index into it which favours the higher ranked trees. In addition to this we use a simple form of elitism in order to preserve the best elements of the population and to give some protection from the bloat (excessive program expansion) that is associated with GP (Poli, McPhee and Vanneschi, 2008). This is implemented by transferring the best two programs from the current to the new generation.

Creation of each new generation of programs is handled through recursive implementations of crossover and mutation in the breed module. Crossover begins with the selection of two ‘parent’ trees at random from the current generation. One of these trees is designated the ‘host’ and the other the ‘donor’. We recursively move down the branches of the host tree, building up the offspring by copying the host. In parallel we recursively move down the branches of the ‘donor’ tree and with a small probability we substitute copies of material from the donor into the offspring we are building up. Mutation begins with a selection of a parent. We then move recursively through the branches of the parent. As we copy the material to create the offspring there is a small probability that a branch from the parent will be replaced with a randomly created sub-tree. It is important to note that only the number of children is required to complete this process. As this is stored explicitly in the Node structure the entire breeding process within the breed module is independent of the operator set.

Once the new generation has been completed the evolve module destroys all programs in the previous generation. This means all storage management is handled explicitly by the
code and there is no need for any automatic memory management (such as a garbage collector) within the framework, a process which can be quite computationally expensive. The cycle of ‘execution, ranking and breeding’ continues until a termination condition is reached. This can be chosen for each system implemented.

To evaluate whether the system was successful we implemented a stock GP example: Finding a mathematical equation. In this a mathematical function is created and using a set of mathematical operators as nodes of GP programs the system is made to recreate the function. We created an operator set of plus, minus, multiplication, greater-than, if-then-else and integers from 0 to 9. We added a fitness function that compared a sample of results from the original mathematical function against the GP programs. The system was successful in discovering the result equation. We compared the system to a tutorial example of the same question programmed in python. We found our system to be on average more than 10 times faster. Our reasoning for this impressive increase of speed would be the use of the more efficient C programming language and the explicit memory management system (Python handles garbage collection automatically).

4.2 Genetic Programming Representation

The initial goal was was to create a genetic programming (GP) system which possessed the properties of sufficiency, type unity and evaluation safety. The first element of this was a suitable representation of the editing programs we would be creating. We needed a representation which could successfully encapsulate the process of editing a video.

4.2.1 Editing as a program

An individual edit of a raw video sequence can be thought of as a series of individual editing operations made on the input video e.g. the cutting of a selected number of frames. It is therefore possible to think of an edit as analogous to a program made up of operations performed on the input sequence.

Developmental Genetic Programming (DGP) describes the process of creating a Genetic Programming framework where the individual programs are used to create an independent output which is the desired result, rather than the programs themselves. This is directly applicable to the idea of an edit as a program and so the idea of creating individual edits through a DGP system was formulated. The system creates programs that do not interpret the measures of interest calculated for the input video nor have any means of program flow. The individual programs create static edits of an input sequence based purely on their individual formulations.
The key development at this point was that of the non-terminal *split* operator. In its initial incarnation it had two children and took as input a video sequence. The *split* operator acts by splitting the sequence in two and passing the sub-sequences on to its child nodes for further processing. We created a simple terminal set comprising *take* and *discard* operators to complete the representation. These two terminal operators take as input a sequence of frames, *take* adding the sequence to the output sequence and *discard* throwing it away.

It is easy to prove the sufficiency of this representation. Taking an unedited sequence of arbitrary length we can, by creating a tree comprising the right arrangement of *split* nodes, split the sequence into its individual constituent frames. We can then create any possible output sequence by applying *take* and *discard* nodes. The system is also type unified.
All operators take as input a frame sequence and have void return values (frames selected by take nodes are appended to a global result). The representation is also completely independent of any information collected from the unedited sequence. This meant that the addition of any new measures to the fitness function would require no changes to the operator set.

As a means of testing this initial system we constructed a sequence generator which would randomly construct binary sequences where a ‘1’ indicates an interesting frame and ‘0’ indicates an empty frame. We constructed a simple fitness function. In order to do this we had to decide on a high level statement to define the requirements of our system, in order for us to encode them into this fitness function. The statement we came up with contained two points. Firstly interest density within the selected footage should be as high as possible and secondly the system should select as much footage as possible. We encoded these two points into our fitness function as two terms. The first took a sum of the ratio of interesting frames to total frames. The second was the sum of selected frames. Each was multiplied by a selected constant and then they were added together. We found it necessary to very carefully fine tune the constants in the fitness function to achieve successful results. Even with a usable fitness function, the results were poor. The resultant sub-sequences very rarely differed in length. We concluded this was to do with the system tending towards a splitting sweet-spot where the system had split the sequence down to a level that, for the individual video clip the selection fitted within the areas of interest. This meant if the areas of interest were large they would only be partially covered. If too short they would not be covered at all.

\[
a * \sum \frac{CapturedInterestingFrames}{SelectedFrames} + b * \sum SelectedFrames \quad (4.1)
\]

We decided simple binary splitting, while being a sufficient representation, would not have enough flexibility to allow the population to evolve adequately. A strict binary representation defines strict boundaries for selections and so an output cut of high value could be formed from concatenation of sub-clips from different branches of the edit program tree. This in itself is not a problem but if the sub-clips’ common parent is high up the tree then
this sequence is very difficult to transfer through crossover. The common parent would
need to be the point of crossover or above and this would include a large chunk of tree with
other sections of less interest being locked in with the area of interest.

Figure 4.5: Problematic selected clip.

4.2.2 Constant nodes

We created the concept of the constant-node as an addition to the tree representation of
programs. A constant-node contains a standardized randomized integer between 0 and the
maximum for a standard C 32 bit integer. Functions were created to convert between this
integer value and a normalized float value between 0 and 1. Constant nodes do not exist
independently; non-terminal and terminal nodes are specified to have a number of child
constant-nodes. When nodes are created their child constant nodes are created with them.

For the split node we added in a constant node representing the split point. This point
defines where the sequence is split; normalized 0.5 indicates divided in two, any higher and
the left segment of the clip will be longer and the right shorter. With this addition a split
node can split a sequence at any point.
One design problem with the constant node is that constant nodes return a different type (integer) to that of the terminal and non-terminal nodes (an edit sequence structure). This breaks the condition of ‘type unity’ meaning we cannot use the process of crossover between constant nodes and terminal/non-terminal nodes. A constrained syntactic structure is a solution to this problem given in (Koza and Poli, 2005). It suggests modifying the crossover and mutation function so that crossover and mutation are separated for each type. We decided that constant nodes are ignored in the tree representation for standard breeding operations; a separate form of mutation was developed for them. Each constant node involved in creating a member of the next generation of edit programs has a small chance of mutating by a random amount within a small number of standard deviations, calculated based on an approximated normal distribution. The split node’s constant can thus be mutated so that the split is itself, in effect, mutated. It is important to note that the implementation of constant nodes is entirely generic within the core parts of the system (creation of and mutation on constant nodes), the only specialization of their use is within the specific genetic operators which have to use the randomized constant value to their needs.

The theory behind the addition of this system was that constant mutation would allow for small improvements in an edit’s fitness value. These would occur as boundaries of splits...
were altered by small increments via the constant mutation. To test the hypothesis that the
addition of constant nodes had improved the system, we tested the system with constant
nodes against the same random sequences of ones and zeros as was used for split without
constant nodes. The same fitness function was utilised as before. The results were much
better than before with the system able to easily select the sections containing interest
(ones) with a much higher success rate.

Figure 4.7: Output from split with constant

Figure 4.7 shows an example output from our system to illustrate the results achieved. The
output is explained by the key but in addition we can see that a continuous line of ones and
zeros indicates a sequence selected by the system. The output shows that almost all 'ones'
have been selected and that areas not containing 'ones' are largely unselected (especially
the large chunk of empty footage at the end). Figure 4.8 shows the improvements in the
fitness function across the generation. There is definite improvement over time. Figure
4.9 shows the standard deviation of population members fitness values. We can see that,
although it varies greatly in the short term, it remains in a fairly constant band over the
long term, which would indicate there is no stagnation of the population as time goes on.
Figure 4.8: Maximum fitness

Figure 4.9: Fitness std. deviation
We discovered a problem with the fitness function[4.1] while evaluating this representation. It became apparent that the two factors in the fitness function were competing with each other to become the dominant factor in the evolution of the program population. This meant the balancing of the coefficients for each factor was problematic. Make the coefficient for the interest density factor too high and this would lead to domination of programs with high density. This would lead very quickly to a program which selects only a single frame containing a 'one' and this would quickly dominate breeding, decimate the population diversity and halt any progression of the population. Similarly if the coefficient for the 'selection amount' was too high then programs selecting large amounts of material would dominate. We would very quickly see a program which selects all material appearing. This would be too dominant to allow any program development in the direction of interest density as this would always cause a drop in 'selection amount' which would drop the fitness more than could be gained by the gain in density. We found the presence of the two local maxima made the process of balancing the two coefficients too sensitive to make the system useful. We deduced that the problem was in how we had defined fitness for our system.

The fitness function of any GP system should encode exactly what the desired result of the system is. We realised that the second term of our fitness function, that of the number of frames selected, was slightly tangential to what we actually required. The desire of our system was to select as much material of interest (frames containing a 1 in this case) as possible while keeping the interest-density high in the material selected. The first term of our fitness function, the ratio of selected interest to number of selected frames, perfectly captures the desire for high interest density. The second term, which measures the total number of selected frames, is slightly mismatched with the actual requirement of selecting as many frames containing interest as possible. By switching this term to measure the number of frames selected of interest we have two terms which perfectly capture the requirements we specified as our high level statement. In addition to this our two terms are no longer completely at odds with each other as they were before. With this system in place we found that the choosing of constants was no longer a knife edge balance and that the range of suitable constants much larger.
4.3 Video sequences

With the system working correctly we began developing a system which would work with video sequences. We had determined that people would be our objects of interest within frames. We tested two possible methods for determining interest within a video sequence: face detection and upper-body detection. We implemented face detection using a Haar-classifier based system utilising the built-in functions of the open-cv computer-vision library. We used an existing 'pre-trained' upper body detection system which utilised a 'Histogram of Ordered Gradients'(Ferrari et al., 2008) based system to detect upper bodies within each frame.
CHAPTER 4. AUTOMATIC EDITING SYSTEM

Testing both systems against a set of sample videos we found that both systems had a more than acceptable detection rate (the ratio of correct detections made to possible detections within a video). We found the face detection had a far higher rate of false detection (where detections were made in error) than upper-body detection. For this reason we decided to utilise the upper body detection system for the fitness function.

The upper body detection measurements were recorded as a rectangular bounding box determining the position and size of the detection. The first stage of designing a suitable fitness function was to decide how this information would be used. We decided that the building block of fitness measurement should be the overlap of the rectangular frame and the upper body detection. The system would be performing the fitness measure against each frame of each sequence created by each edit program in the population. It was therefore imperative we created an efficient means of calculating it in order that the system could execute in an acceptable time period. We developed an efficient algorithm to calculate any overlap of frame and detection. \[4.3\] shows my rectangle representation and the calculation of the overlap in the x dimension (it is only calculated when a overlap-defining boolean expression evaluates true). It simply places the x positions of the two testing rectangles into a 4 element array and then sorts them using the ‘c standard library’ quicksort function. The overlap is then the difference between the middle elements of the sorted array.

Listing 4.3: Rectangle overlap calculation

```
typedef struct rectangle *RECTANGLE;
struct rectangle {
    int x1;
    int x2;
```
```c
int y1;
int y2;
}

// Calculation of x dimension overlap
int xOverlap;
int sides[4] = { r1->x1, r1->x2, r2->x1, r2->x2);
qsort(sides, 4, sizeof(int), compare_ints);
xOverlap = sides[2] - sides[1];
```

With this measurement defined we incorporated it into our two term fitness where we substitute in the rectangular overlap as our measure of interest. To this basic fitness function we could now add extra measures. (Arijon, 1991) and (Spottiswoode, 1935) both allude to the fact that extremely short cuts within a sequence are unpleasant for the viewer (in some situations they can be used, such as a sequence of fast action shots, but these are far too specific for a general home video edit.) In order to incorporate this we included an extra term in the fitness function which was a coefficient multiplied penalty term for sequences which contained cuts of less than a set minimum frame limit (initially set to 30 frames).

\[
a \cdot \sum CI + b \cdot \sum CI/FrameCount + c \cdot ShortClipsCount
\]  

(4.2)

With this function (CI is captured interest) set down we prepared an initial video for testing. A sequence involving a young child posing for the camera and then running off camera was used as it contained a long sequence at the end of empty uninteresting footage along with some shaky footage at the beginning. The upper body detector made most detections that would be expected against a ground truth. As a first test we ran against a basic system containing only split, take and discard operators. The results were excellent. Figure 4.12 shows an example result sequence. Judged on an aesthetic scale the system had picked out three cuts from within the high interest footage of the raw sequence. Figure 4.13 shows the evolution of the maximum fitness value and shows good development over the generations. The standard deviation remained noisy but within a constant set of bounds throughout the run.

![Figure 4.12: A representation of the initial edit program](image-url)
4.4 Cropping

The next phase involved the development of a system to crop footage. We added a cropping node to the terminal set. It has four constant nodes representing the top left and bottom right corners of the rectangle representing the portion of the screen selected by the crop. Thus a crop node takes a sequence passed to it as input and returns a selected sequence which has been cropped. We envisaged that for an output video would take any cropped footage and expand it to the correct frame size. In this way cropping would act in a similar manner to the digital-zoom functionality of a common digital camera.

This new feature required a redevelopment of a new output representation to take account of the variable footage size. We built a list structure which contains the start and end frame numbers along with the ‘RECTANGLE’ structure which holds the frame selected. The list element of the structure allows me to append different sequences together to form the complete edit. In order to describe the output of a ‘take’ node we simply use the rectangle which describes the full frame.

Listing 4.4: Example code

```c
struct RectSeqList {
    int startFrameNumber;
    int endFrameNumber;
};
```
One drawback of the first iteration of crop representation is the four constant nodes. This represents a large parameter space for each crop node. As it is contained within 1 node it cannot be altered by the primary method of evolution: crossover. All improvements in cropping must be done through constant mutation and thus the number of constants must be kept to a minimum. The four constant system also contains a fair amount of redundant parameter space defining frames too small or of unsuitable aspect ratios. To ensure that the range of all possible crops and to streamline the parameter space we redefined the system with only three constant nodes:

- A scale constant from nought to one. Scaling from half length/width to full frame selection.
- X and Y position constants representing the top left corner of the scaled frame. The effect of these constants are dependant on the scale value as for full frame selection these constants are redundant and scaled frames (0,0) would indicate the top left corner being in the top left corner of the overall frame and (1,1) placing the bottom right hand corner of the scaled output in the bottom right hand corner of the overall frame.

This representation has distinct advantages over and above the reduction in the number of parameters. Any crop within the range of possible crops is now definitely of a useful shape and of the same aspect ratio as the original footage. This comes at the slight disadvantage that crops of non standard ratio that are still of an acceptable shape once expanded (not overly warped) are not available.
We evaluated *crop* against the sample video used for testing *take* and *discard*. Results were aesthetically not very pleasing. Figure 4.15 shows a sample result where a series of *take* and *crop* sequences are taken together. If we look at the graph of fitness value over time in figure 4.16 we can see that development is very sharp to begin with but after only a short number of generations, growth halts. We decided this could be a problem with the population becoming stifled.
4.5 Panning

In order to represent the idea of a moving camera within a frame we developed a panning operator. We formulated panning as an extension of cropping. A panning sequence is represented as the cropped frames at the beginning and end along with the frame numbers at the start and end. Thus intermediate cropped frames which describe the pan can be calculated using simple linear interpolation of the frames sizes as illustrated in 4.5. To describe this within my tree representation we designed a terminal ‘pan’ node. It had six constant nodes the first three representing the crop of the beginning of the sequence and the final three the end crop.
Thus a ‘pan’ node will take as input a start and end frame number and output a sequence containing these frame numbers along with the start and end rectangles of the pan. In order to facilitate this within my output representation we changed the single crop rectangle into start and end rectangles as shown in 4.5. This meant a change to the way a crop was represented. A ‘crop’ can be considered a pan where the start and end frames are the same. Thus we use this representation. In a recursive manner a take is therefore a pan where the start and end frames are the same and the maximum size.

Listing 4.5: Example code

```c
struct _RectSeqList {
    int startFrameNumber;
    int endFrameNumber;
    RECTANGLE startFrame;
    RECTANGLE endFrame;
    struct _RectSeqList *next;
};
```

As we tested this system against our initial set of videos we were unable to get any meaningful pans to form part of the output videos. Where a pan did appear in output videos it was in no way beneficial to the framing of persons within the frame. The conclusion we drew was that the six constant nodes contained in a pan node described a parameter
space that was far to large to be workable. As these parameters are all within a single terminal node there is no way for crossover to have any action in the evolution of pans so all optimisation would have to be through constant mutation which is not strong enough to work in a parameter space of six dimensions.

We noticed that the single terminal node pan representation contained two distinct and independent elements: start and end rectangles. These rectangles use the same three constant system (scale factor and x/y offset) as used in ‘crop’ nodes. We hypothesised that if we could make these rectangle representations interchangeable between crop and pan then crossover between these operators would occur. Opening up crossover as a means of population evolution would then greatly improve the systems ability to create useful pan operations, the thinking being that crop is a much simpler operator and thus there would be suitable crops in the representation that, through crossover, would be able to evolve into pans. The key concept is the interchangeability of the rectangle representation between crop and pan.

We switched to a non terminal representation for pan with two children, the idea being that the node would receive the start and end rectangles from its two children. The node would take as its input start and end frame values. It would pass on an arbitrary 1 frame sequence to its children to receive as simple a sequence as possible as a in return. The representation would receive sequences from its left and right children and select the start rectangle from each to represent the start and end values from the pan.
To test whether this representation was indeed an improvement we devised a simple test. We created a simple test data set emulating the motion of a person across the centre of the screen in a sequence. While a large number of different good results would be possible, among them a pan following the moving body would be expected. We used the same simple fitness function attempting to capture as fully as possible within each frame the simulated upper body detection.

\[ a \sum \text{CapturedInterest} + b \sum \frac{\text{CapturedInterest}}{\text{SelectedFrameCount}} \]  

(4.3)

'a' and 'b' are constants. We tested both representations of pan, non-terminal and terminal, against the same fitness function and test video and determined whether either representation produced any pans, and whether or not any of these pans successfully captured the footage.
The video results were not too encouraging. Neither set of results were aesthetically pleasing. Both seemed to be following the simulated detection across the screen but the pans seemed erratic. We noticed that non terminal followed with some success but was not always capturing the entirety of the detection box. It was also noticeable in Figure 4.19 that all increases in fitness occurred in the very early generations and that both systems peaked at almost equal fitness levels. This could suggest a problem with the fitness function. Looking at the data for the standard deviation of the population fitness there did not seem to be any problems with lack of variety as the variation was remaining noisy in a constant range. Also there is no variation in the frame size selected by the videos, it is always the smallest frame size allowed by cropping. The obvious reason for this would be the nature of the test video, which contains only a small simulated detection, but it could hint at a problem with the evolution.

4.6 Conclusion

We have created an excellent efficient genetic programming framework. We have created a novel genetic programming representation of the editing process. The split constant representation works extremely well. The results for crop and pan operators are inconclusive and will require further evaluation.
Chapter 5

Evaluation

To evaluate the system we produced a set of test videos which were chosen to cover a wide range of (slightly contrived) situations. Each was processed by the upper-body detection software and the resulting detection data was passed to our editing systems. We then analyzed the output. All videos in this section are listed in Appendix 5.1 Split

We carried out the majority of testing on our core system: split, take and discard as our representation. We use as our fitness measure the area of screen covered by upper-body-detection bounding-boxes. The fitness function chosen was:

\[ 0.25 \times \text{AreaCovered} + \frac{\text{AreaCovered}}{\text{FrameCount}} \]  

(5.1)

We simplified the short clip penalty given for aesthetically unpleasant short cuts. We decided the penalty for programs creating output sequences containing clips shorter than 30 frames was a zero fitness score.

5.1.1 Video 1

The first input video we created has an empty start and end. The centre section involves a person walking onto screen, waving and then leaving. The detector recorded near flawlessly; the protagonist was detected in almost all frames in which he is almost front on to the camera. There were very few false positives.
CHAPTER 5. EVALUATION

The editing system outputs a video sequence containing the central chunk of footage containing the person, cutting out the empty start and end footage. This is what we expected from the system. The max fitness value graph shows that there is massive early improvement followed by some gentler adjustments of the fitness value. We concluded the first phase involved finding the shape of the general solution with the second phase involving minor adjustments of that solution. The graph of fitness standard deviation is very noisy and so it is difficult to draw any conclusions.
We include an example output tree representation for this edit along with a representation of the actual frames chosen (the remainder are in Appendix C). In the output representation '1' indicates a chosen frame containing detection, 0 indicates a chosen empty frame, a space indicates an empty unselected frame and a period indicates an unselected frame containing a detection. The sequence is read line-by-line, left-to-right. The lines of stars are the upper and lower boundaries of the sequence with the ‘—’ symbol marking the exact end of the sequence.

Listing 5.1: Example result tree

```
split
..0.800738
..split
....0.870548
.....discard
.....discard
..split
....0.501968
.....split
......0.837557
......split
........0.374407
........discard
........take
```
5.1.2 Video 2

We created the second video with capturing multiple persons in mind, and also to determine whether the system could function where it is the majority of footage which should be selected. The upper body detection is again near flawless.

The system’s solution is what would be expected; the vast majority of detections are selected for the output footage. The graph of max fitness shows the solution to this simple situation was arrived at very quickly as the max fitness reaches a plateau in less than 10 generations. Fitness standard deviation is again large and noisy.
CHAPTER 5. EVALUATION

Figure 5.5: Fitness value

Figure 5.6: Standard deviation
5.1.3 Video 3

We devised the third video to be more challenging to the system as it contains a sitting subject and a lot of camera movement. The close range of the shots allows the detector to operate at a very high success level.

![Diagram of Video 3 output]

The system captures the three periods in which the subject is captured in footage. In the output video the nature of the camera movement and the sequences the edit has put together means there are sharp changes in motion between cuts which creates a disorientating effect. It would be appropriate if measures could be put in place to discourage this kind of cutting. Fitness and standard deviation graphs show very similar results to those of the previous videos.
CHAPTER 5. EVALUATION

Figure 5.8: Fitness value

Figure 5.9: Standard deviation
5.1.4 Video 4

We chose an outdoor location to test the capability of the system to deal with different conditions and a greater level of movement. The detector was able to detect the majority of persons but seemed to struggle with subjects further than close range.

![Video 4 output](image)

The selected footage is aesthetically very pleasing and the periods containing good detection are selected. The graph of max fitness shows two periods of growth before reaching a final plateau.

![Fitness value](image)
5.1.5 Video 5

We chose a longer and more challenging video sequence with no completely clear sections of a subject standing directly in front of the camera. The upper-body-detector had more difficulty making detections and therefore the detections are spread sparser in the footage.

The system selected very little of the footage and while the footage selected is of a high quality the output video is very short. The max fitness and standard deviation graphs are in line with those of the other videos.
Figure 5.14: Fitness value

Figure 5.15: Standard deviation
CHAPTER 5. EVALUATION

We decided to see if increasing the constant 0.25 factor, which the ‘AreaCovered’ fitst term is multiplied by, would increase the amount of footage selected. We slowly increased the value to see whether the selection increase could be achieved. When the constant was raised to 0.625 the system selected a much larger portion of the footage without selecting large empty periods. The video created by this altered fitness function is of a much higher quality, with clear watchable cuts.

![Figure 5.16: Video 5 output with adjusted fitness constants](image)

5.1.6 Conclusions on Split

The core system works extremely well, making good decisions about which footage is to be selected depending on the presence of upper-body-detections. It seems to cope well with varied densities of detections within the sequences. The only slight drawback is that has been shown that the balance between overall detection and detection density in the fitness function may need to be altered, in certain situations, to get the desired results.
CHAPTER 5. EVALUATION

5.2 Pan and Crop

Figure 5.17: An output frame with accompanying detection (blue) and cropped frame (red). The portion in red indicates the chosen portion of the screen.

We tested the non terminal pan representation against the first video from our set. The fitness function was the same as before except that the ‘proportion of overlap’ (between detection and frame) replaced the area covered by detection as the central measure of fitness.

Figure 5.18: An output sequence for the non-terminal representation of the pan operator.

As can be seen the results were not successful. There was very little correlation between the upper-body-detections and the positioning in the frame. Thus any output video would be unwatchable if rendered completely from the selected frame positions. The maximum fitness function trace suggests that any advancement of fitness in the population is finished after only twenty generations. Similarly poor results were achieved with the other videos in the sequence and we decided to end formal testing on this section.
5.2.1 Conclusions on Pan and Crop

We concluded that there is a problem with either the fitness function or the representation. It could well be that a parameter space of three constant nodes, which cannot take part in crossover, is too large for simple mutation to allow useful development. It could also be a problem in the fitness function whereby the programs are reaching minor peaks of fitness which do not allow development of fitness to continue as any changes in the area of this minor peak will mean drop in fitness and therefore a less successful program with less chance to influence the next generation.

5.3 Conclusions

We found that the evaluation has lead to two strong findings. Firstly, the basic representation utilising split, take and discard operators is strong. We have achieved excellent results. The upper-body-detector software gives strong clear detection and the system is capable of exploiting this to select clips containing as many detections as possible. One slight drawback is that if the detections are slightly less successful and sparsely spread (as happens in more active video clips) then the constant ratios in the fitness function require modification. The second finding is that the crop/pan representation is not yet successful.
Chapter 6

Conclusion

The project has contained equal measures of success and failure. On the one hand we have fulfilled the core deliverable; we have created a system which utilises a strong measure of interest (upper body detection) and implemented a genetic programming (GP) framework to edit videos. On the other hand, the extensions attempted were not successful.

We have found a novel application for GP, and with the *split*, *take* and *discard* operators we have created a successful novel GP representation of the editing process. The evaluation of this representation found it to be extremely successful and it managed to create excellent edits of each test video we used.

Tempering this success however are the failures of the *pan* and *crop* operators. In theory they fit well with base set of operators and do link in nicely with them. In practice we could not implement them to edit the input videos successfully. One possible explanation for this would be the lack of variation in our mutation and crossover methods which could be constraining the development of the population. A further consideration is that three constants on a single *crop* node could strain the random constant mutations ability to improve the genetic programs.

6.1 Conclusions on Implementation

The implementation of the project was a strong point. The use of the C programming language meant the system was extremely quick and efficient. We also designed the core system to be independent of any operator set used with it meaning that no additional development was required on the core when implementing new operators. This allowed experimentation with operators and lead to the non-terminal representation of *pan*. While this was not successful it was an improvement on the original terminal (six constant node carrying!) representation.
6.2 Conclusions on Project Planning

One of the drawbacks in using a low level language like C, as opposed to a high level language, would be that development can take longer as low level considerations (such as memory allocation) must be taken into account. We believe it was a mistake to leave serious system development of the core system until after Christmas. We spent the time before this on experimentation with side issues such as detection methods. This left a serious amount of pressure in the post exam period and lead to lessening the scope of the interest measures we could develop.

6.3 Further work

The project gives great scope for extension due to the open ended and subjective nature of aesthetic video editing. We have left many possible avenues of expansion.

6.3.1 Fitness measures

We have created a genetic programming representation which, due to its independence from measurements taken from a video sequence, can act as a solid base for the development of a more sophisticated fitness function. We considered many further fitness measures we could add to the system.

Optical flow (Beauchemin and Barron, 1995) can be used to measure the relative motion of selected ‘features’ (parts of the image which are readily identifiable) in a video sequence. There are numerous possible applications:

- We could use optical flow measurements from the whole image to estimate the camera motion during the sequence. This could be used to build in a term in the fitness function which penalizes cuts that match conflicting camera motion ((Arijon, 1991) states that if the direction of camera motion is changed sharply then it can have a negative effect on the viewer, as was the case in one of the result videos).

- We could consider areas of optical flow independent of the camera motion as indicating interesting motion and reward frames that capture this interest.

- We could measure the optical flow within upper body detection bounding boxes which could indicate motion of the detected person. We could then attach a reward for a certain suitable capture of this motion, such as zooming in close.

The inclusion of measurements of sound would add another aspect to our measurement of interest. Periods of high volume levels relative to the average level could indicate periods of interest. We could also consider some measure of vocal activity detection, as used in
(Hua et al., 2005), and then reward the capture of vocal activity or highly penalize any cutting off of periods of vocal activity.

We also concluded that a system which could judge the quality of the sequencing of camera movements within an output would be an interesting avenue to explore. We could model different aesthetic concerns with differing fitness function terms.

6.3.2 Genetic programming operations

A genetic programming system is always open to extension and improvement. A number of possible improvement the evolutionary capabilities of our system arose during development but were left out due to time constraints. The first was the idea of seeding in the higher level structures (such as take and crop nodes) in the form of the low-level pan constructions (as an example a take would be created as a pan with identical ‘full frame’ start and end rectangles). This would allow seeded take and crop nodes to make good base points to evolve from (as ordinary take and crop nodes cannot evolve into pan constructions.

Further improvements could include a variety of additional mutation operators, such as a flip operator which would alter the operator type of a particular node without disturbing the structure of the program tree, could greatly improve the breeding process. This and many other mutation operators are suggested in (Poli, Langdon and McPhee, 2008).
Bibliography


Koza, J. R. (1990), ‘Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems.’.


Appendix A

Project Plan

Overleaf is my project gannt chart.
Figure A.1: Gantt chart for the project
Appendix B

Video Results

On the accompanying CD we provide a selection of sample result videos.

B.1 Development

initialInput.avi, initialOutput1-2.3gp: The initial test video and results.
cropResult1-3.avi: Some crop results for the same footage.
TpanResult1-5.avi, NTpanResult1-3.avi: Test results against a simulated detection. Note: These videos are taken against a background of a sample video of which the results bear no relation. It was simpler to do this than create blank frames.

B.2 Evaluation

input1-5.avi, output1-5.avi: Standard input and output results.
output5.avi: Extra test against modified fitness function (longer footage).
resultNTPAN1.avi, resultTPAN1.avi: Evaluation pan results.
Appendix C

Raw results output

These represent the raw solutions given by the editing system for the videos in the evaluation chapter.

Listing C.1: Video 1 output

******************************************************************************

  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111
  111111111111111111111111111111111111111111111111111111111111111111111

******************************************************************************

Listing C.2: Video 2 output

******************************************************************************

  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111
  .1111111111111111111111111111111111111111111111111111111111111111111111

******************************************************************************

Listing C.3: Video 3 output

******************************************************************************
Appendix D

Code

The majority of code is on the accompanying disc as it is too extensive to list here. I include the file evolve.c which drives the process.
D.1 File: evolve.c

//EVOLVE_C_

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <signal.h>

#include "nodes.h"
#include "init_tree.h"
#include "rank.h"
#include "rand.h"
#include "breed.h"

#include "print_tree.h"
#include "debug.h"
#include "g_params.h"

NODE pop[2][G_POP_SIZE];
struct rank ranks[G_POP_SIZE];

void clear_gen(int gen);

//Used for clean exits
sig_atomic_t siggy = 1;
void quitHandler(int sig);

//send stats to file
void record_pop_stats(int);

int evolve(void)
{
    int gen = 0;
    int gen_no = 0;
    int current_best = 0;

    signal(SIGINT, quitHandler);

    //initialize the first population
    init_pop(pop[gen], G_POP_SIZE);

    do
{  
  // rank the population
  rank_pop(pop[gen], ranks, G_POP_SIZE);
  // record population statistics
  record_pop_stats(gen_no);
  ++gen_no;
  if (gen_no == 1 || ranks[0].score < current_best) {
    printf("\n\n\nGeneration%5d:%8d\n", gen_no, ranks[0].score);
    fflush(stdout);
    current_best = ranks[0].score;
  }  
  else if (gen_no % 100 == 0) {
    putc('.', stdout);
    fflush(stdout);
  }
  // Switch generations
  gen = (gen+1)%2;
  // Free all nodes in spent generation
  clear_gen(gen);
  // Two best selected
  pop[gen][0] = copy_tree(ranks[0].tree);
  pop[gen][1] = copy_tree(ranks[1].tree);
  // Breed the rest except ...
  int i;
  int parent1;
  int parent2;
  NODE temp;
  for (i = 2; i < G_POP_SIZE; i++) {
    if (rand_norm() > G_PROB_NEW) {
      parent1 = (int)(log(rand_norm()) / log(G_SEL_COEFF)) % G_POP_SIZE;
      parent2 = (int)(log(rand_norm()) / log(G_SEL_COEFF)) % G_POP_SIZE;
      // breed...
    }
  }
}
parent2 = (int)(log(rand_norm()) / log(G_SEL_COEFF)) % G_POP_SIZE;
temp = cross(ranks[parent1].tree, ranks[parent2].tree);
pop[gen][i] = mutate(temp);
free_tree(temp);
}
else // ... add in a few random trees for variety
{
    pop[gen][i] = random_tree(G_NEWTREE_LGTH, G_NEWTREE_MIN);
}
}
}
while(gen_no < G_MAX_GEN && siggy);

// print out best score
printf("%END_SCORE%d\n", ranks[0].score);
// print out the winning tree
print_tree(ranks[0].tree);
// deal with any required output
rank_result(ranks[0].tree);

// Empty populations
clear_gen(0);
clear_gen(1);
}

void clear_gen(int gen)
{
    // Free all nodes in spent generation
    DEBUG(INTO CLEAR GEN
    int i;
    for(i = 0; i < G_POP_SIZE; i++)
    {
        free_tree(pop[gen][i]);
    }
    DEBUG(OUT CLEAR GEN
}

void quitHandler(int sig)
{
    siggy = 0;
}

void record_pop_stats(int gen_no)
{
FILE *file;
int i, max, total;
double average, sdtotal, diff, sd, size = (double)G_POP_SIZE;
// record max value
max = -ranks[0].score;
// record sd
total = 0;
for(i = 0; i < G_POP_SIZE; i++)
{
    total += ranks[i].score;
}
average = ((double)total)/size;
sdtotal = 0;
for(i = 0; i < G_POP_SIZE; i++)
{
    diff = ((double)ranks[i].score) - average;
    diff *= diff;
    sdtotal += diff;
}
sd = sqrt(sdtotal/size);

// write to file/
file = fopen("./gnuplotResults.txt","a");
fprintf(file,"%d %d %8.2lf\n", gen_no, max, sd);
fclose(file);
}