VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

An Investigation of Primitive Set Selection and Evaluation for Image Generation Using Genetic Programming

Roman Klapaukh

Supervisors: Mengjie Zhang, Will Browne

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

In genetic programming for evolutionary art, there are many primitive sets in use. Often their capabilities are unknown. This project defines four primitive sets and compares their abilities to generate a number of images. Overall it finds that a primitive set based on drawing shapes performed the best. However, none of the primitive sets managed to generate images with sufficiently high quality.

Acknowledgements

I wish to thank my supervisors, Meng and Will, who have had to put up with me all year. Similarly, I would like to thank the entire evolutionary computation research group for their feedback on my work, as well as suggestions, advice and help.

My family have also supported me through this year. It is always nice to have somewhere to go home to, especially after you start feeling like you might be spending almost all of your time in a computer lab.

I would like to thank everyone in the Spongebob community. To Sophie, who has put up with me despite all the stressing all year, Michael, Jan, Sam, Gus, Cat, Josh, Liam, Bruce, Amy, Adam, Vicky, Daniel and Jing who have all made Spongebob what it is today.

I thank everyone who has put in time to proof read this report, especially Carlton who spent a lot of time helping me. Without them and their red pens, this report would be a horrible mess, full of spelling mistake and general rubbish.

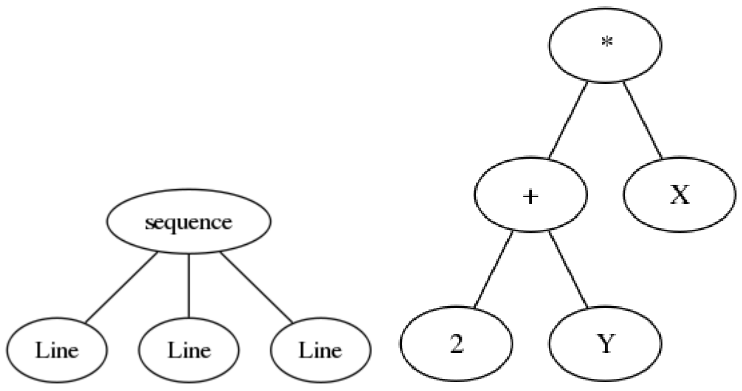
Contents

Chapter 1

Introduction

Evolutionary art seeks to recreate the human ability to create art. Dorin [11] asks whether this is reasonable. If an image is created by searching through the space of all images until a "desirable" one is found, is the creator skilled? Are they even an artist? Taking Dickie's definition of art, "a work of art is an object of which someone has said, 'I christen this a work of art'." [10], it is easy to label the output of such programs as art. However, this is no indication of quality. Art can be understood in the context of what meaning or history it has, and what feelings it evokes [26]. Yet art produced by computers may have no such history unless there is a human to guide it, or computer programs with emotions can be created. By creating tools for the purpose of creating evolutionary art many authors have taken a side in this argument. Cook's paper [7] asserts that the images it produces are art. Similarly the NEvAR (Neuro Evolutionary Art) system [20], by virtue of its name, implies it is expected to produce art. However, these systems are not built on solid foundations. All questions of aesthetic quality aside, the range of images these systems can generate is not clear. It is unknown whether an arbitrary image can be recreated with any of these systems. This is not to say that an evolutionary art system has to be capable of generating any given image, just as oil painters do not have to produce pencil sketches or sculptures to be called artists. However, the range of producible images should be defined in advance. This project explores the range of images that can be generated when using genetic programming, a form of evolutionary computation, for this task.

Evolutionary computation is useful because it can generate novel solutions. In evolutionary computation a way of representing solutions is required, as well as a *fitness function*, a function that is capable of evaluating the quality of a solution. There are three main approaches to fitness functions for this task. The first is having a human in the loop [29, 12, 7, 20]. In this method, a human is used to evaluate all images produced and give feedback to the system. The second approach involves comparing potential solutions to an image or set of images known to be good [3, 23]. The initial set of good images is provided by a human expert. The third approach is to judge the quality of images solely using an algorithm [24, 9, 19, 27, 15]. Hybrid systems also exist, such as in [28, 22, 30]. Using a human to evaluate images is effective as this system uses a fitness function that corresponds to the users tastes. However, as the user is performing all evaluations, there are several significant problems. The number of images evaluated by each user, has to be limited to prevent user boredom. Moreover, the time taken for the user to react and judge images is a bottleneck. Using a set of good images as a reference removes the need for constant repetitive user input, as the selection of good images only has to be done once (possibly over an extended period of time). However, it is not certain that the system will be able to identify the right attributes needed to judge the fitness of unseen images accurately. The main advantage of using an automated system is that it requires no interaction with an expert or any human



(a) Sequence representation (b) Mathematical representation

Figure 1.1: GP programs

to produce results, thus overcoming the identified limitation. However, it does require an algorithm that is accurately capable of evaluating how aesthetically pleasing a given image is, which is still an open problem.

Within Evolutionary Art, Genetic Programming (GP) is a technique that is used to address the problem [4, 16, 18, 20]. GP generates tree structured representations of programs. Different trees can be combined to create new and potentially better trees, and any tree can be used to create an image. GP uses *Terminal* and *Function* nodes to represent the trees. Terminal nodes are the leaves of the tree, while function nodes are the internal nodes. The *function set* is the set of all functions available to the GP system. Similarly the *terminal set* is the set of all terminals available. The *primitive set* is the set of all functions and terminals. For the purposes of this project a distinction between the function set and terminal set is unnecessary, as they only form a complete representation together. An attractive feature of GP for developing new systems is that it allows for primitive sets to be easily changed, allowing GP to express a wide variety of solutions. In GP it is important that a primitive set is sufficient, that some combination of the nodes are able to express an optimal solution. For evolutionary art we require something stronger, we require additionally that the primitive set will be likely to converge to a good solution within reasonable time.

Despite the variety of approaches and systems for evolving images there do not appear to be many distinct classes of primitive sets, at least for the GP systems. There are two main types of GP primitive sets that are considered here. The first is using *Sequence* nodes [4, 16] as in Figure 1.1a. Sequence nodes do not draw, they only give the tree its shape. The terminals are the actual drawing operations, such as *Rectangle* or *Line* nodes. This gives program trees that represent a sequence of drawing instructions, and such trees are easy to understand. The second type of representation is using standard mathematical functions as in Figure 1.1b. If needed, the output value is scaled, and turned into a pixel colour value [29, 20, 18]. Using shapes to model images can be difficult as they are often not made up of simple shapes. Hence it may take large numbers of simple shapes just to make up one small complicated section. The mathematical representation has the advantage of being frequently used in other commonly seen problems such as symbolic regression, but it is very hard for a human to understand the resulting programs. In these works the reasoning behind the choice of primitive set, is often not explained.

1.1 Goals and Objectives

This goal of this project is to investigate the choice of primitive sets for use with GP in evolutionary art. Variations of the *Sequence* node represenation will be compared with *mathematical representations* with the aim of determining if there is a difference in their expressiveness. Specifically this project has three main objectives.

1. To create four primitive sets for use in GP for the creation of images. The four primitive sets are:
   - Single colour lines — This primitive set is a simple version of the colour shapes one. Its main purpose is to provide some insight into the different possible ways of structuring trees comprised of drawing operations. Having a set with only one drawing operation should make the comparisons easier. Three different tree structures will be explored.
   - Single colour shapes — This primitive set is to represent drawing using basic shapes. Similar primitive sets have been used successfully [4, 16], and this is an intuitive way to approach drawing as a human can understand it. This is the method that is hypothesised to perform the best, as a decomposition of an image into shapes does not seem very complicated for humans.
   - Mathematical functions — Primitive sets like this have been very commonly used [25, 29, 20, 18] and as such are important to include in the comparison. This method is not hypothesised to produce results as good as the single colour shapes, but is anticipated to produce recognisable results. Unfortunately it is hard to get a good handle on how the mathematical functions will behave as this representation does not match up with how people perceive images.
   - Self normalising mathematical functions — This is the same as above except that multiple strategies are used for converting numbers into colours. It is anticipated to perform similarly to the mathematical functions.
2. To compare the four primitive sets on how closely they can copy a target image.
3. To determine which of these primitive sets produces the closest copy, and examine its suitability for generating arbitrary images.

1.2 Major Contributions

This thesis has made the following major contributions.

1. This project shows the design of four different primitive sets that can be used in GP for generating images.
2. This project shows the differences in the expressive abilities of the various primitive sets. We find that it is non trivial to create a primitive set to copy an image, due to their inherent complexity. It is not enough to just have similar colours in the correct places. Colours have to form actual shapes with distinct boundaries, and in many images, with intricate details. Finding programs that place the correct colours at the correct co-ordinates as in the target image has a low probability. Moreover, the interaction of two side by side lines of slightly different colours in the generated image, can appear as a boundary or detail to a human, even if it should not. This results in the primitive sets producing different stylized copies of the original.

3. This project shows that the shapes primitive set has the best performance on the images that were used in this work. Despite this, it may not make a good primitive set for use in other work. This is due to its tendency to place objects near, but not exactly, where they should be. While this produces images that are recognisable copies of images, they are still quite different from the original.

1.3 Organisation

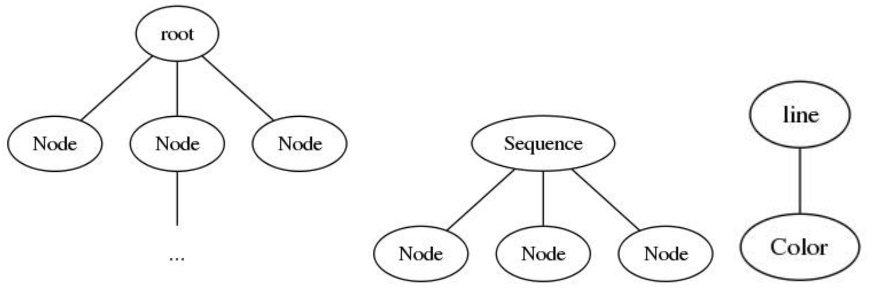
The remainder of this report is organised as follows. Chapter 2 contains the background needed to understand the work done, as well as the motivations for this project. Chapter 3 describes the details of all primitive sets considered in this thesis as well as the criteria for judging representation quality. Chapter 4 describes the results of all the experiments. Finally chapter 5 has a discussion of the results and chapter 6 gives the conclusions and future work.

Chapter 2

Background

2.1 Genetic Programming

Genetic programming (GP) is a machine learning technique that uses selection pressure and a mating process to produce solutions [17]. This has many parallels with evolution and genetics. In genetic programming, as it is used in this project, a program is represented by a rooted undirected tree (Figure 2.1a). Each node in the tree is either a *function* node or a *terminal* node. A function node is a node which has children such as the *sequence* node in Figure 2.1b, and a terminal node is a leaf of the tree, like the *colour* node in Figure 2.1c. The way that this tree is parsed and the operations performed by each of the nodes results in the output program produced. In genetic programming the choice of functions and terminals is very important. If a solution cannot be formed from possible combinations of the functions and terminals given, then GP will never be able to find the solution to the problem.



(a) GP tree (b) Sequence node (c) Line node

Figure 2.1: Genetic programming components

For genetic programming it is common to define a fixed *population size* (number of programs in each generation) for the duration of the run. A set of genetic operators is also defined. This project uses the following three genetic operators: elitism, mutation and crossover, which will be discussed in the next section. Each of the three operators is used to create a fixed proportion of the next generation.

2.1.1 Genetic Operators

Elitism selects the top possible individuals in each generation. This ensures that the best individual of the next generation is at least as good as the best of the current generation. As this operator is the only one that has no random element to it, it is the only one that is certain to produce individuals that are good by the standards of the current generation. However, while having not enough elitism may result in a lot of poor individuals being selected for the next generation, having it set high may prevent the solutions from getting better as it can only provide ones that already exist.

Mutation selects a random node in the program tree of a particular individual, deletes it, and replaces it with a random, valid subtree. This adds in new genetic material, and may introduce new permutations of nodes into the gene pool. This also allows for the refinement of small areas of the tree. In the biological metaphor this can be compared to random damage to DNA due to random effects such as radiation.

Crossover selects a random subtree in each of two parents and swaps them, similarly to how two parents contribute to the genetic make-up of a child. This has the potential to combine the good parts of two solutions into a new better solution. This does not always happen, so it is used to explore the space near the given solutions.

Both mutation and crossover act by replacing parts of program trees, so the structure of the tree becomes important. If the program is structured such that any attempt to remove a part of the tree would destroy its value as a good solution, then this sort of program representation may not lead to good offspring being produced by crossover. As both operators work by selecting random subtrees, neither is capable of dealing with the situation where an individual is very good except for having a poor root node. If mutation were to change the root node it would also then have to change the entire tree, losing the good parts. Crossover on the other hand would have to select each of the root node's children in turn and swap them into a tree with the correct root node. On top of this, as the program is structured as a tree, the majority of the nodes are at the bottom of the tree. As such when a random node is selected, it is more likely to be one of the nodes at the bottom of the tree. For this reason, representations in which all the computation is done at the leaves are more likely to be modified in useful ways than ones where the computation is done inside the tree.

2.1.2 Fitness

The other key component of genetic programming is the fitness function. This is a function that can take an individual program and assign to it a value describing how "good" it is. Goodness is not a concrete measure, but rather is a way of judging how close a program is to the perfect program. For example, in the case of evolving programs to add numbers, a possible fitness function could be the difference between a given program's answers and the real answer. This fitness value is then used for selection, where better programs (programs that have a better fitness) are more likely to be selected for breeding to populate the next generation. At the end of the GP process the individual with the best fitness is the solution. The selection of a good fitness function is critical to the success of GP as a technique. If the fitness function is not quite right, then the GP process will not be attempting to produce a good program for the true problem, and so will be unable to produce good results.

2.1.3 Selection

A selection mechanism called Roulette Wheel Selection [14] (also known as Proportional Selection) is used for all parts of this project. In Roulette Wheel Selection programs are selected with a probability directly proportional to their fitness. An individual may be selected multiple times. This mechanism is good as all individuals have some chance of being selected for use in the next generation.

An alternative selection mechanism is called Tournament Selection [14]. This requires a parameter called *tournament size*. Here, a number of individuals (based on the tournament size) are sampled, and the one with the highest fitness is selected as the breeding individual. The main problem with this is that the least fit individuals will not get selected (as some other individual in the tournament will have a better fitness) but they may have characteristics that are worth keeping.

2.1.4 Initial Population

At the start of the GP run the initial population is generated randomly. There are three mechanisms for accomplishing this.

- The first of these is *Grow* where programs are just generated by picking random elements from the primitive set until a valid tree is created.
- The second is *Full*. Here programs are generated randomly, but they are forced to be as deep as the experimental setup allows.
- The third is *Ramped Half-Half* where half of the individuals are generated using the Full method and half using the Grow method.

2.1.5 Generating Solutions

Using the above tools the genetic programming process is then run:

1. Generate an initial population
2. Evaluate each individual
3. Select individuals for breeding
4. Breed the individuals to create a new population of the same size using the genetic operators
5. Until you reach the maximum number of generations or find and acceptable solution go to step 2

2.2 Evolutionary Art

Evolutionary art started out with Dawkins evolving little plant programs [8]. However, his work did not focus on the idea of art or creating images, so much as showing that evolution is a good model. Sims' work [25] however was explicitly targeted at creating aesthetically pleasing images / textures using mathematical operations.

Since then much more work has been done and there are now two main approaches. The first is using a human as the fitness function [29, 12, 25, 7, 20]. Images are shown to a human user who determines how good an image it is. The second approach uses a computer to evaluate the fitness in one of two ways. The first uses an expert to predetermine a set of "good" images that can then be used as the basis for any future work. This can be done with learned classifiers [3] or by comparing to certain properties of the images [23]. The second method avoids all user interaction [24, 9, 19, 27, 15, 2, 5, 1]. Using a human to select an initial set of images still needs an expert to pick the initial set of images. Also the system will have to have either a fixed target, or need to learn a new classifier each time a new image is required. Even in the case of a fixed target, it is often not sufficient to reproduce the target image (as it must already be available to be used as a target and generating a copy of an image is usually not the goal). Fully automated systems do not have any bottlenecks due to human interaction and are capable of generating new images transparently. However they require some form of algorithm capable of judging the fitness of an arbitrary image, a still open problem.

Using humans as the fitness function removes the problem of having to find a way to algorithmically detect how aesthetically pleasing an image was, but introduces new problems such as user-fatigue, where the user starts getting bored [30]. This can lead to needing small populations and short runs [29], or large numbers of users [12]. Trying to use known good images as a basis for creating new ones is additionally difficult. First it requires a human expert to create the set of known good images, which, while being potentially time consuming only needs to be done once, and so is reasonable. However, it is important that the correct properties of the image be used as a guide, which is also difficult [23]. Systems that use automation for all fitness evaluations are able to be more general, and run without any human users. However, the problem of judging the aesthetic quality of an image algorithmically has no current good solutions, and so these systems can only have very limited use.

These can further be broken down into systems that use shapes to draw images, and those that use mathematical functions. Systems that used mathematical functions for their representation [25, 29, 12, 20, 23, 3] are the majority based on the papers surveyed. While there are systems that use shapes such as GAUGUIN [7], they are the minority. In general, the choice of primitive set was not explained in the papers, and there was no guarantee of their ability to produce the desired results. The only interesting exception to this is the Electric Sheep system [12]. This system explicitly uses flame fractals for all their images (a fixed mathematical representation where only the arguments are changed), however unlike most systems, Electric Sheep produces movies. While the flame fractals it uses are not capable of modelling any given image, they do not need to be for this application. They are simply creating movies that are aesthetically pleasing to users of their application (which run as screen savers), and so their representation does not need to be able to draw arbitrary images. It has been shown , through running the system for a time, that aesthetically pleasing movies can be produced. It is possible that the other systems also ran some kind of tests to see if interesting images were ever produced by their systems before finalising the primitive sets, but this is not stated.

Systems that use automated fitness functions are the most affected by choice of primitive set. Ross et al. [23] attempted to get traits from current artworks to create a fitness function. They used the same representation as in Sims' work [25], but the expressive capacity of that representation was not evaluated or considered. Hence it is unclear if their system is capable of producing images that are similar to their original set of good images. This limits their findings as it is unclear how much effect the primitive set has on the final images produced, and as such it is hard to generalise their results to arbitrary representations. For similar reasons all experiments testing fitness functions have more limited results than necessary. While these systems were evaluated on how well they managed to produce images, there was no reason to expect the systems to be able to produce the desired images. The works done by Barile et al. [5] and Alsing [1] are exceptions in this. The work of Barile et al. only seeks to stylistically copy existing images. It also uses information about the image that it is trying to copy in the drawing process, and so has an advantage in drawing images as it already knows what it should produce and will not take counter productive steps. Alsing's work specifically set out to copy the Mona Lisa using 50 polygons with a fixed representation, and as such is sure in advance that the representation is what is desired. Another similarity is that both these works only copy images of faces. This may be significant as images of a face have a certain structure, as most human faces are similar. As such, while these works managed to produce good copies of the original image, they did not test a variety of images, and in the case of Barile et al., used information about the target image that would not be available to a system trying to draw a new image from scratch.

Between all these works a variety of primitive sets are used, and many interesting images are produced, but there was lack of good justification for the particular primitive sets chosen. This carries with it a risk of misjudging the fitness functions (for the experiments that were testing out new fitness functions), and deciding that the fitness function is a poor judge of images when the real reason the images look poor is that the representation is unable to produce images that that fitness function would rank very highly. While the systems created work, produce images, and facilitate research on automated fitness functions, the selection of a primitive set is not something that has had a lot of attention. This project focuses on providing some comparisons of different primitive sets. This will provide more information about systems that attempt to evolve images, as now it is unknown if they are capable of evolving arbitrary images, or only certain kinds of images.

Chapter 3

Method

The main focus of this project is to determine the expressiveness of different primitive sets. To that end, four primitive sets are defined and compared to each other. The four representations are based on previous works. A large area of research in evolutionary art is the design of a fitness function capable of judging the aesthetic quality of an image. Such fitness functions are intrinsically linked to the primitive set used, as it serves as a constraint on the search space. By comparing these four primitive sets, this project aims to provide additional context in which to evaluate fitness functions.

Two of the four primitive sets represent images using mathematical expressions, and the other two use shapes. Experiments were run to compare the ability of these four representations to copy various target images. Experiments were used as opposed to mathematical analysis since this is difficult due to the non-deterministic nature of GP. An ideal representation will encourage the formation of good solutions. A good solution is one that produces an image that is very similar to the original. In the case where the means are not significantly different, the representation with the lowest standard deviation from the mean will be taken as the best. The low standard deviation is desirable as it means that the representation produces more consistent and predictable results.

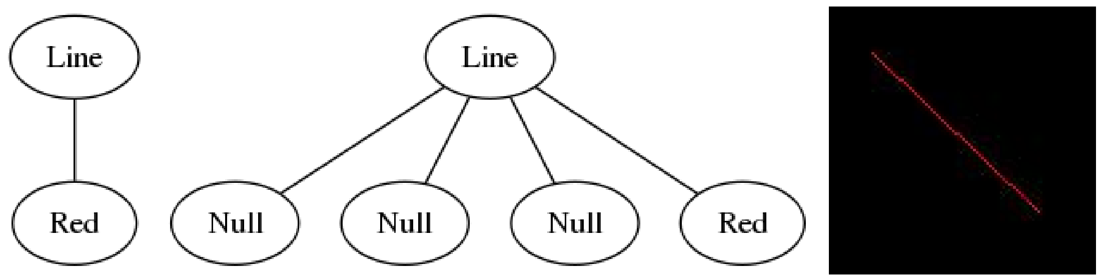
3.1 Representations

All of these representations use RGB colours. This means that each colour is comprised of three integers each in the range [0..255], where (0,0,0) is black and (255,255,255) is white. While other colour representations exist (such as HSL or indexed colour), the RGB representation is used in this work as it is simple, sufficiently expressive and there exist useful programming libraries for it.

3.1.1 Lines

There are many different ways that a sequence of instructions to draw lines can be represented. Due to also needing to test representations other than lines, this project only considers three different representations for drawing sequences of lines. These representations are three adaptations of the representation used by Barile et al. [4]. In this representation *Sequence* nodes are the functions (Figure 2.1b). The sequence nodes have multiple children but do not do any computation themselves. *Line* nodes have children representing their own position and their colour.

In the first representation computation is performed at the bottom of the tree. Sequence nodes have varying numbers of children and lines have their position as an internal property and their colour as a child. This version is hypothesised to perform the best, as it has all the



(a) Line with no children (b) Line with Null children (c) The resulting image

Figure 3.1: Example red lines image

nodes that have side effects at the bottom of the tree, so it will be easier for useful nodes to be modified by the GP operators, which struggle to make changes to the middle of a tree.

The second version moves computation to the inside of the tree. In this representation a Line has a colour child and its position is stored internally. These lines also have three other Line children. There are also *Null* nodes. These can take the place of a Line and do not do anything as in Figure 3.1b. The Null nodes are just there so that the program tree can terminate.

The third version has computation done in all parts of the tree. Lines are as in the previous representation, but Null nodes have been replaced with Lines with no children as in Figure 3.1a. This is so that all nodes in the program tree contribute to the image. As these representations have the drawing nodes inside the tree, it is hypothesised that GP will find it harder to modify individuals in ways that increase their fitness and so they are not anticipated to perform as well.

Both lines and shapes share the same possible representations. Due to time constraints, comparing all the shapes representations as well as all the lines representations was infeasible. As such the simplifying assumption was made that the best shapes representation would be based on the best lines representation.

3.1.2 Shapes

The shapes representation is an extension of the lines representation. Shapes use the lines representation except that instead of only having *Line* nodes, they also have circles and polygons with three to five sides (Figure 3.2). Polygons have been restricted to five sides to reduce complexity. It is anticipated that polygons with more than five sides can be approximated by other operators. Many sided convex polygons can be approximated by circles, and concave polygons by multiple simpler polygons.

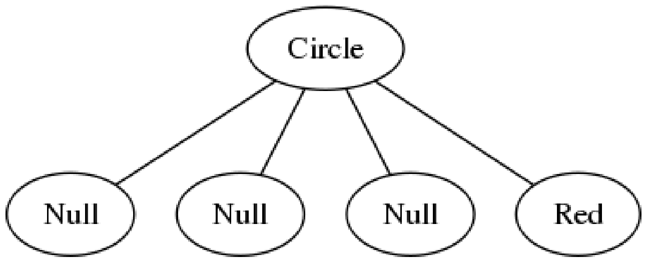


Figure 3.2: Example red circle

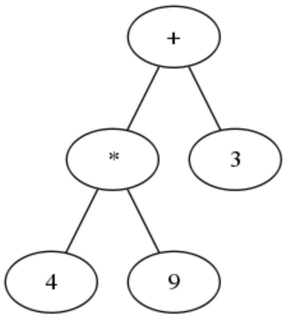


Figure 3.3: $(4 * 9) + 3$ as a tree

3.1.3 Mathematical Functions

Mathematical expressions already have a standard representation as a tree as in Figure 3.3, and as such this same representation was used in this work. The mathematical functions used are a set of standard mathematical functions over doubles, such as addition and multiplication, with the full list appearing later in this section. Each colour channel has a separate tree to represent it. This has the unfortunate side effect of making each colour channel in effect a separate image. As such it is possible to get images that look like the colours are all unrelated to each other, as they have no relation in the program tree. However it also allows each colour channel to be independent, and hence able to represent certain kinds of colour transitions more easily.

Arbitrary mathematical functions over the real numbers can return results that are real numbers, and may not be valid colour values. As such the final result of the computation is put through a function to constrain the range (set of possible outputs) to be the integers from 0 to 255. Functions used for this purpose will be referred to as *normalising functions* [2]. There are however, many different functions that can do this.

The distinguishing feature of this mathematical representation is that only one normalising function is used for all images. The normalising function is outside of the evolved program, reducing the search space. The normalising function used here is the modulo function shown in equation 3.1 and Figure 3.4a.

The mathematical functions used are addition, subtraction, division, multiplication, logical and, logical or, logical xor, sin, arctan, exp, log, min and max. There are also nodes X and Y which sample the x and y co-ordinate respectively, as well as *RandDouble* which produces a random double value. While this is not the full set of functions used in other works, this set contains most of the commonly used functions, and seems to be similar to the functions used in the IMAGENE system [29]. As such any conclusions drawn from using this function set may not apply directly to any other primitive set. A limited primitive set was chosen as there is no standard set of mathematical functions used, and this set contains the set of well known functions. It is hoped that results from using this set will provide a baseline that can be used for evaluating other mathematical representations.

3.1.4 Normalised Mathematical Functions

This representation is the same as the mathematical functions representation except that the normalising function is a function in the GP tree. This is different from the previous mathematical representation as the normalising function is now a part of the GP process. This results in a bigger search space, as any of the three normalising functions can now be used, and potentially more interesting images. The three functions described in section 3.1.5 will be the possible normalising functions that can be used. This representation is hypothesised to produce better results than the plain mathematical functions representation, because the normalising function can affect the way the colours are drawn, as discussed in the next section.

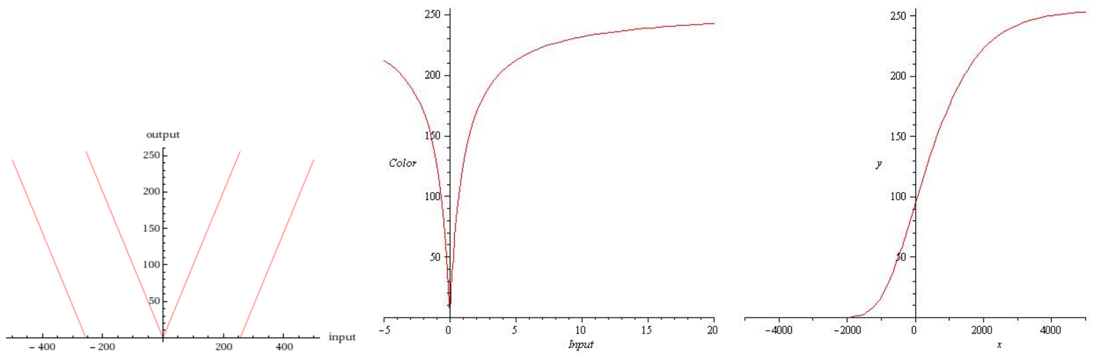
3.1.5 Normalising Functions

Normalising functions [2] are functions that accept arbitrary real values and return an integer in [0..255]. This project uses the three functions shown in equations 3.1, 3.2, and 3.3. Each of the functions behaves in a quite different fashion. Equation 3.1, as in Figure 3.4a increases steadily until it reaches 255 then drops suddenly back to zero, only to repeat. Equation 3.2, as in Figure 3.4b is close to 255 when it is far from zero, and drops rapidly to zero the closer the input gets to zero. On the other hand equation 3.3, as in Figure 3.4c gets larger as the input value increases, while never going outside of the allowed range. How fast it grows and where the steep part of the curve is, depend on the parameters *b* and *d* which are evolved as part of the image generation.

$$N(c) = ||\lfloor c \rfloor|| \ \% \ 256 \tag{3.1}$$

$$N(c) = \left|\left| \left\lfloor \frac{255c}{c+1} \right\rfloor \right|\right| \tag{3.2}$$

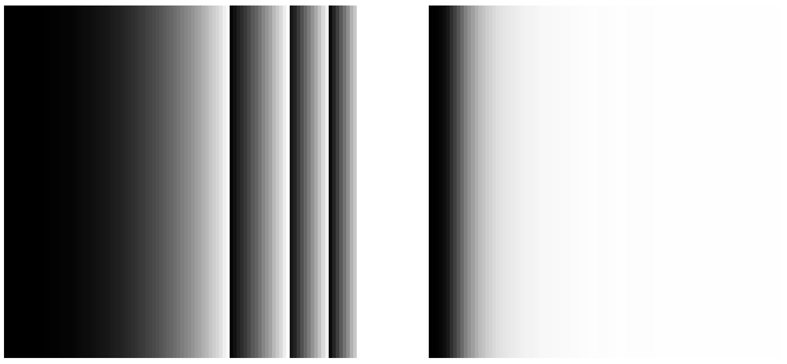
$$N(c) = \left\lfloor 255e^{be^{dc}} \right\rfloor \tag{3.3}$$



(a) Modulo function (b) Ratio function (c) Gompertz curve

Figure 3.4: Normalising functions

Normalising functions are not all the same. Normalising functions change how an image looks. Consider a GP tree for a grey scale image which is of the function X^3. This tree will give a higher value result the further right the pixel is, starting with 0 at the left. Hence, it would be anticipated that the image will be black at the left and become lighter to the right. This is indeed the case if a normalising function such as the ratio function (equation 3.2) is used, as can be seen in Figure 3.5b. However, if the modulo function (equation 3.1) is used then, moving to the right, it gets gradually whiter, but suddenly becomes black again, and then starts getting white again, as in Figure 3.5a.


(a) X^3 using modulo normalising function

(b) X^3 using ratio normalising function

Figure 3.5: Effect of normalising functions showing two of the three for comparison purposes

3.2 Fitness Function

The fitness of an individual is measured by the distance from the original image. This is the sum of the absolute differences in activation for each pixel in each colour channel. Let an activation for a pixel in an image be $image_{colour}[x][y]$, the target image be t, and an image produced by the GP algorithm be o. Then the fitness of o can be expressed as in equation 3.4. For the GP system that is used here, fitness values range from 0 to 1, where 1 is the best. So equation 3.5 is used to turn the fitness into a transformed fitness value for use in this system. It divides the fitness by the highest possible error, where each pixel has the maximum deviation possible from the target, for each of the colour channels, then subtracts that from one to make 1 the best fitness.

$$\text{fitness}(o) = \sum_{x=1}^{width} \sum_{y=1}^{height} \sum_{colour} |t_{colour}[x][y] - o_{colour}[x][y]| \tag{3.4}$$

$$\text{scaled fitness}(o) = 1 - \frac{\text{fitness}(o)}{3 * 256 * \text{height} * \text{width}} \tag{3.5}$$

This is not an ideal fitness function, even for copying an image. This fitness function only looks at the individual colour of each pixel. It does not factor in concepts such as the relative locations of pixels. This means that images can have high fitness using the fitness function, but still not resemble the original image to a human. This is because all the pixels in the image may be close in colour to the target, but the image as a whole may not have any forms that resemble objects. As such a person will not see anything similar to the target image, but the algorithm will give the image a high fitness. However, an image with sufficiently high fitness will have to resemble the original image strongly, as there is no other way to have all the pixel values be close enough. Hence, based on the assumption that all our representations will work at least reasonably well, and the produced images have sufficiently high fitness, this fitness function will be adequate for this project.

Limits

Given this fitness function, a bound is needed to decide how high the fitness must be for an image to be considered a sufficiently good copy. RGB can represent $256^3 = 16,777,216$

colours. The human eye can see on the order of 2 million colours [21], and we assume that colour perception is the same over all colours (for simplicity). Therefore RGB can represent many more colours than a human can distinguish. Hence we will allow a small amount of latitude in what constitutes a good copy.

Let n be the number of different values you need for each colour channel such that you have $2,000,000$ colours total. Hence

$$\begin{aligned} n^3 &= 2 \times 10^6 \\ n &= \sqrt[3]{2 \times 10^6} \\ n &= 126 \text{ (0 dp)} \end{aligned} \tag{3.6}$$

As n is about half of 256, that allows a colour value in a colour channel to be one value off. This amounts to a total unscaled fitness of 5 per pixel (one channel is three values off, and the other two are just one value off). Given that all images used in this work are 100 by 100 that gives us a final fitness of $5 \cdot 100 \cdot 100 = 50,000$ for an image. We will now define a convincing copy of an image as one with a fitness at least as high as an image where each pixel is two perceptible colours off. This corresponds to an unscaled fitness of 70,000 (an upper bound value, when one channel is five values off and the other two are only one off) which is a scaled fitness of 0.99089 (5 dp). This is to allow the system some leeway to make mistakes, but still requiring it to produce accurate copies of the target image.

3.2.1 Limitations

The conclusions reached in this thesis may not hold if a different fitness function is used. It is possible that using a different fitness function could lead to a different evaluation of the primitive sets. However, in the present analysis it is assumed that our fitness function can be used as a representative of fitness functions in general. We also note that here we are using the fitness function to copy a particular image, whereas the representation is intended for use in evolving new and novel images. It is assumed that copying is an "easier" task than creating something new, and as such a representation which is inadequate for copying an image will be similarly inadequate for evolving an original image.

Despite these problems, if a representation is not capable of representing an image, then it is not capable of representing it irrespective of what fitness function is used. If a representation is not capable of generating certain kinds of images, then when a fitness function is being evaluated or used, even if the fitness function favours the images that cannot be represented using the given primitive set, good images will not be generated.

The images that are being copied in this work are real art works, and not simplified synthetic examples. The reason for this is that the experiments themselves take a very long time, and there were only seven months allotted for this project. As such, it was worth running the more complex images, as there would have been no chance to test them otherwise.

3.3 Selecting a Lines Representation

An experiment was carried out to compare the three lines representations, to determine which is the best, so that it can subsequently be used for tests, and function as the basis for designing a shapes representation. Each of the representations was used in more than 30 runs (the exact numbers vary as some of the runs were terminated by other users of the computing system, and could not be restarted due to time constraints) to copy a target image. The target image is Figure 3.7, a large extract from The Voyage of Youth - Life by

Parameter	Value
Population	100
Generations	200,000
Elitism Rate	2%
Mutation	48%
Crossover	50%
Max Depth	8
Image Size	100 x 100
Selection	Roulette Wheel
Program Generation	Ramped Half-Half

Figure 3.6: Experimental settings for comparing lines

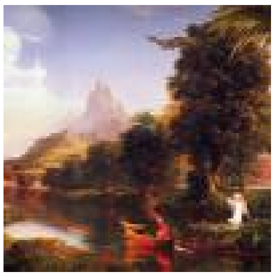


Figure 3.7: Target image for lines

Thomas Cole. This image was chosen as it is an image that is widely accepted as art (in the Western world at least) as well as being in the public domain. While the chosen image is not the only possible result that could be a desired output, it is still desirable, and time constraints did not allow for having a wider sample of art. For this project, a desirable primitive set should be able to produce this image. If it cannot produce this image, then it does not matter if it can produce others, as we are seeking a general use primitive set.

In this part of the project the RMITGP framework [13] was used to do the genetic programming. The RMITGP package was used to reduce programming errors as most of the implementation had already been done, and as it written in C++, to get faster code execution.

The configuration for the GP system can be seen in figure 3.6. A roughly equal mix of crossover and mutation was chosen on the basis that the crossover would provide the general structure of the image, and mutation would “fix up” small details. As such both of these were needed in large measures, so a roughly equal split was used. Elitism is at 2% as it was desired to keep the best images from each generation, so that each generation would be at least as good as the one before. It was assumed that each generation would be quite poor, so there was no reason to keep too many images. A population size of 100 was used as some empirical tests run showed that larger populations did not seem to give better results. 200,000 generations were used as there seemed to be a lot of scope for improvement in images that were run for smaller numbers of generations (e.g. 100,000), and it was the largest generation value that could have been used given the time allowed for an honours project. It is also a sufficiently large number of generations that it is unlikely that systems will be set to run for significantly more. An image size of 100 x 100 pixels was used as anything larger made the fitness calculations take unreasonably long. Roulette Wheel selection and Ramped Half-Half program generation were chosen to give programs with poor fitness that may have some good features a better chance to survive, with the goal that they may be able to provide some details to images with better fitness. A max depth of 8 was chosen to give 2,187 drawing nodes for the representation with the least number of line nodes. This was assumed to be enough given that the image only had 10,000 pixels. Test runs had also shown that the produced trees did not tend to use all possible nodes, and setting it any higher made the experiments run for too long.

3.4 Comparing the Four Representations

Once the best lines representation was selected, an experiment was carried out to compare the four image representations focused on in this work. The most successful lines representation was used in this section, as well as a shapes primitive set based on its structure. Both mathematical representations were also used. All four representations were set up to copy three images. For each image 13 experiments were run using each representation. As this experiment was carried out over a distributed computing grid, it was uncertain that all the experiments would be able to complete successfully and so the largest number of experiments it was practical to run were used. The images used are sections from old artworks that are out of copyright. Three art works were chosen as that was the most that there was time and resources to do. They are: The Voyage of Life (Youth) by Thomas Cole (Figure 3.8a), Child with Red Hair Reading by Lilla Cabot Perry (Figure 3.8b) and a statue of Leonardo da Vinci by Luigi Pampaloni (Figure 3.8c). Fitness will be evaluated as described in section 3.2.

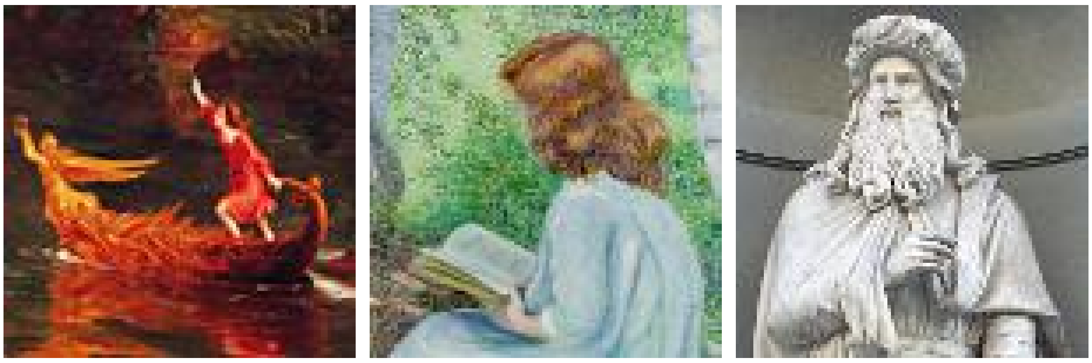

(a) Voyage of Life - Youth (b) Child with Red Hair Reading (c) Statue of da Vinci

Figure 3.8: Target images

3.4.1 Experimental Set up for Comparing the Four Representations

All experiments were run using a Java translation of the RMITGP package. The package has been modified to allow program trees with multiple roots, in order to allow for the sorts of trees used in the mathematical representations. The translation into Java was needed to facilitate using the grid as the machine used for testing the experiments ran a different operating system and hardware from the grid machines.

The configuration for the experiments can be seen in figure 3.9. We use almost the same setup as in section 3.3 with the same reasoning. In the results of the lines experiments (in chapter 4), it was noted that none of the fitnesses were close to the target fitness desired. One possible reason for this is that many good images were being lost from the solution pool due to poor children being created, and the elitism being too low. As such the elitism rate was increased. Images using shapes representations have natural optimal substructure. Optimal substructure means that an optimal solution is comprised of optimal solutions to subproblems. In the context of images, a perfect image is composed of perfect subimages. As such, a good part of each image could be evolved in separate programs, and then combined by crossover. In order to take advantage of this property, crossover was increased. Mutation was decreased in order to facilitate these changes.

Parameter	Value
Population	100
Generations	200,000
Elitism Rate	5%
Mutation	28%
Crossover	67%
Max Depth	8
Image Size	100 x 100
Selection	Roulette Wheel
Program Generation	Ramped Half-Half

Figure 3.9: Experimental settings for comparing the four representations

Chapter 4

Results

4.1 Significance Testing

For all significance testing in this work, the rank sum test at 95% significance (using MATLAB v7.10.0.499(R2010a) 32-bit (win32)) was used. This test was used instead of the t-test as the underlying distribution is not assumed to be normal.

4.2 Selecting a Lines Representation

In the experiments to compare the different lines representations, not all the experiments successfully terminated. This was because they were all run on separate computers, and some were turned off or reset while the experiments were running, as the computers were remote from the experimenter. The experiments could not be restarted due to time constraints. A summarised version of the results can be seen in table 4.1, and the complete results can be seen in table 4.2.

Considering only the means, it would appear that the representation that used Null nodes performed the best. Significance testing showed that the representation that used only lines (No Null in the tables) has a lower mean than the Sequence node and Null node terminated representation. The other two however, do not have significantly different means. The Sequence node representation has a much higher standard deviation than the Null node representation, as can be seen in table 4.1 and Figure 4.1. Figure 4.1 shows the distribution of final fitnesses for the Sequence node and Null node representations. The lines-only representation is omitted as all its fitness values are too low to be seen on the graph. Figure 4.1 shows that the final fitness values for the Sequence node representation are very spread out compared to the Null node representation. For the subsequent experiments, some experiments may not terminate, as with the lines experiments. Having all the final fitness values close together will give a better indication as to the mean fitness of the representation.

Overall the Null node representation was chosen as the best representation as it provided a high mean value, but also consistently produced low variance results.

	Mean	Std. dev.
Sequence	0.86239	0.02494
No Null	0.71744	0.01271
Null	0.87242	0.00705

Table 4.1: Lines final fitness summary

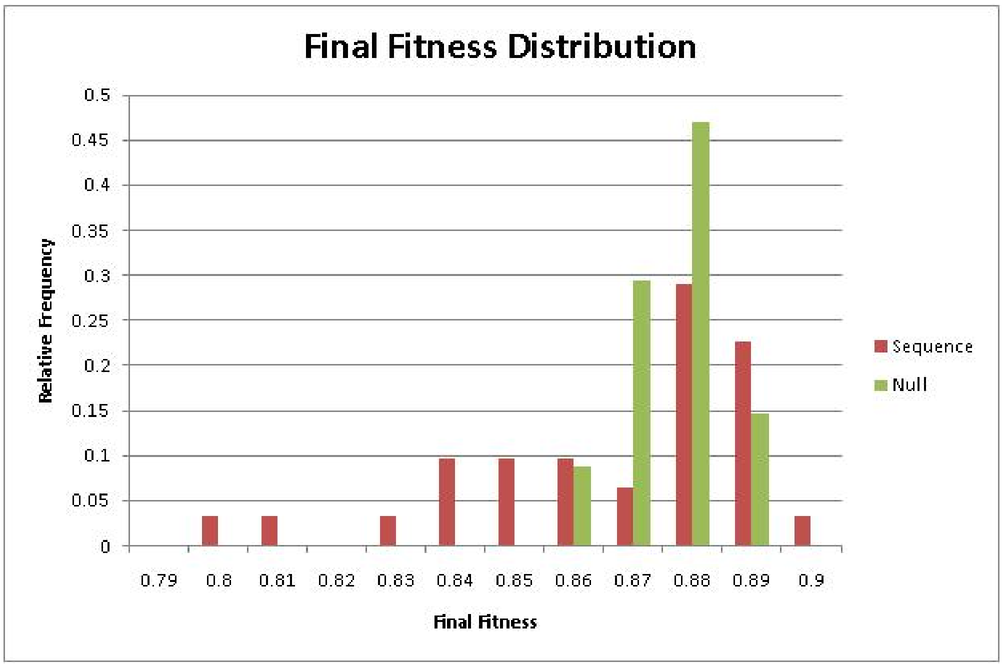


Figure 4.1: Final fitness distributions for Lines

Sequence Nodes		No Null		Null	
0.79443	0.87619	0.71448	0.71548	0.85749	0.87367
0.80782	0.87714	0.74512	0.718	0.85983	0.87403
0.82812	0.87769	0.74221	0.70778	0.85986	0.87405
0.83003	0.8781	0.70939	0.71772	0.86134	0.87475
0.83241	0.87919	0.71251	0.71422	0.8655	0.87589
0.83786	0.88062	0.70512	0.74172	0.86587	0.87671
0.8449	0.8847	0.716	0.72122	0.86737	0.87702
0.84706	0.88515	0.71122	0.70686	0.86738	0.87725
0.84883	0.88546	0.71422	0.72105	0.86798	0.87726
0.85233	0.88721	0.74986	0.71158	0.86822	0.87766
0.85278	0.88742	0.70162	0.7121	0.86878	0.87845
0.85617	0.8883	0.71262	0.71268	0.86906	0.88069
0.86554	0.89453	0.70513	0.71511	0.86962	0.88135
0.86814	0.87138	0.7189	0.709	0.87174	0.88173
0.87061	0.87332	0.71846	0.70301	0.87224	0.88424
0.87072		0.74681	0.70857	0.87233	0.88708
		0.71278	0.72784	0.87252	0.87317
		0.71008			

Table 4.2: Final fitnesses for Lines

4.3 Comparing the Four Representations

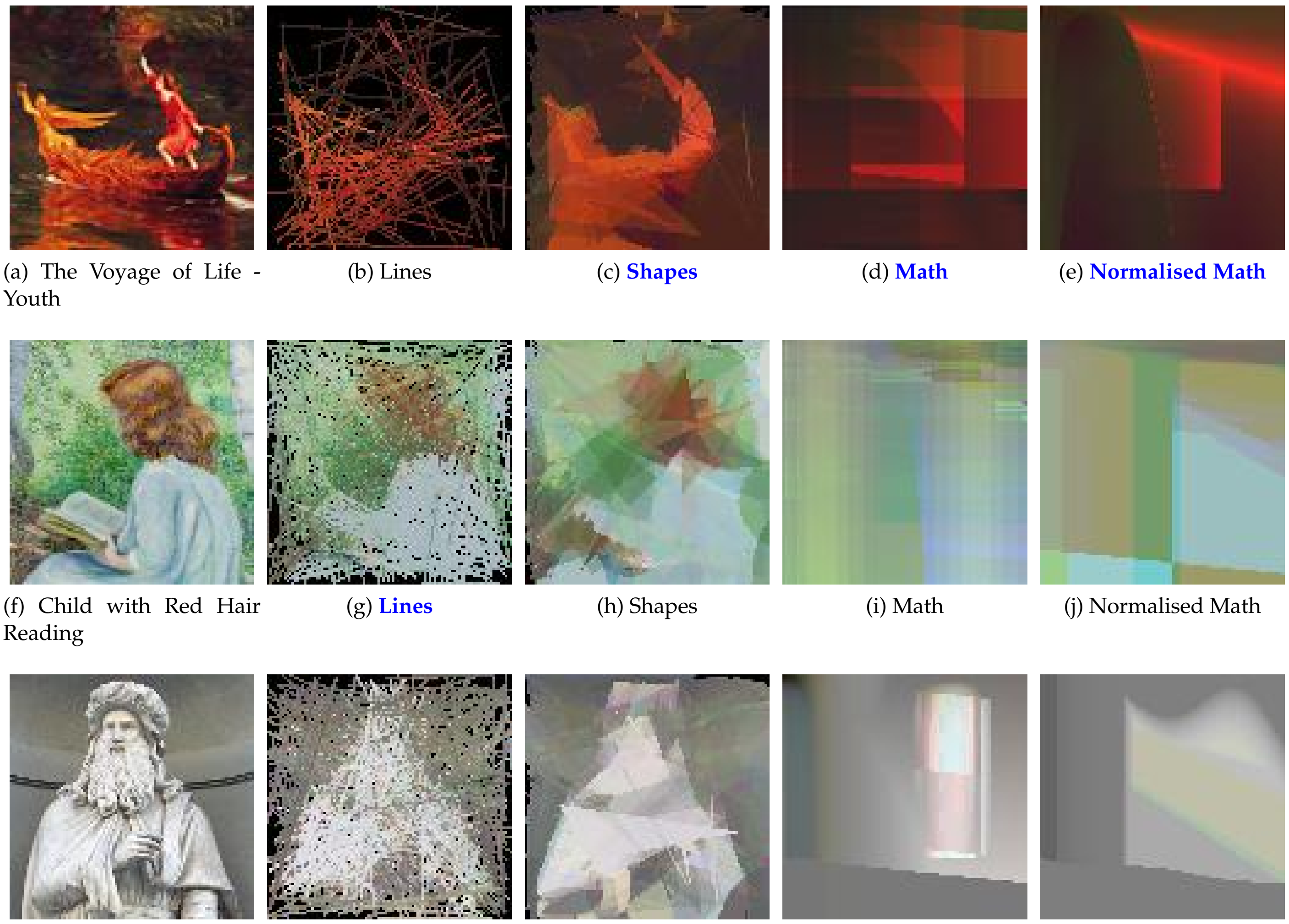
All the final fitnesses can be found in table 4.3. There are gaps in the table where experiments did not finish. This is due to the non-deterministic nature of grid computing. Originally we forecast that experiments would take a month to finish, using liberal assumptions. These assumptions did not take into account several factors we were unaware of, and the data in table 4.3 is taken after over two months of running experiments. The means were calculated over all three images, as we are not interested in the performance on an individual image. The results (with significance testing) show that the lines representation performed the worst and the shapes performed the best. The two mathematical representations did not have significantly different means, and performance was between that of shapes and lines.

The best image in terms of fitness, from each representation for each target image can be seen in Figure 4.2, where the best image from a given representation has its caption in blue and bold (the reader is encouraged to look at the generated images from a distance). There are two key things that can be seen from these images. The first is that for all but the lines representation, the best image produced was The Voyage of Life - Youth (Figure 3.8a). The second is that the images generated by the mathematical representations look nothing like the target image, and yet according to the fitness function used, the mathematical representations preformed better than the lines one. This is despite the mathematical images being considered a worse copy when using a human to evaluate the fitness, as in the human-in-the-loop technique, they are considered better copies.

None of the images had fitness values of 0.99089 or over to rank as a convincing copy of an image (as defined in section 3.2). Despite this, the shapes representation preformed the best, and has images that resemble the target image, particularly when seen from a distance.

	Lines			Math			Shapes			Normalised Math		
	child	daVinci	life	child	daVinci	life	child	daVinci	life	child	daVinci	life
1		0.84576	0.84009	0.87308	0.84823	0.91417		0.91005		0.86793	0.87066	0.90417
2	0.83709	0.84652	0.84287	0.87096	0.85915	0.89915	0.91588	0.91098	0.91987	0.86797	0.85974	0.9072
3	0.84861		0.84073	0.87111	0.88115	0.91501	0.91721		0.91522	0.87211	0.85668	0.907
4			0.84516	0.85765	0.8453	0.91444			0.92483	0.87653	0.86058	0.90348
5	0.84513	0.8416	0.83856	0.81429	0.84552	0.90942		0.91162		0.88397	0.85121	0.91695
6	0.83247	0.84905	0.84797	0.85588	0.8349	0.91198			0.92097	0.86966	0.85924	0.91016
7		0.84763	0.84691	0.8696	0.84748	0.91176	0.91275	0.91274	0.92552	0.88033	0.85462	0.90983
8	0.83402			0.87118	0.85983	0.90709			0.91354	0.87964	0.86156	0.91407
9		0.83795	0.84654	0.85773	0.84339	0.89524	0.91273	0.91591	0.92522	0.86792	0.8533	0.91197
10	0.84934			0.86404	0.85064	0.91177	0.9155	0.90374		0.86919	0.86377	0.91373
11	0.8531	0.8413	0.83667	0.86121	0.85255	0.91076			0.92283	0.86894	0.87401	0.90711
12	0.83728		0.83802	0.87482	0.87084	0.9129		0.91106	0.9201	0.87149	0.86518	0.908
13		0.84837		0.8639	0.83801	0.90127		0.91158	0.92372	0.87308	0.86261	0.91093
Mean		0.84303			0.87429			0.91624			0.88119	
Std. dev.		0.0054			0.02779			0.00576			0.02158	

Table 4.3: Final Fitnesses for All Representations



(a) The Voyage of Life - Youth (b) Lines (c) **Shapes** (d) **Math** (e) **Normalised Math**

(f) Child with Red Hair Reading (g) **Lines** (h) Shapes (i) Math (j) Normalised Math

(k) Statue of da Vinci (l) Lines (m) Shapes (n) Math (o) Normalised Math

Figure 4.2: Best Image From Each Representation for Each Target Image

Chapter 5

Discussion

5.1 Selecting a Lines Representation

The results show that the Null node representation gave the best performance. Both the Sequence node and Null node representations for lines performed equally well considering their mean results. However the Sequence node representation generated results with a much larger spread of fitness values and hence a larger standard deviation. For the purposes of this experiment a lower standard deviation is good as it helps to produce consistent and predictable results.

The spread of results in the Sequence node version is quite interesting. Before starting the experiments, it was hypothesised that the sequence node version would perform the best, as it allows for all drawing nodes in the tree to be changed easily, since they are always on the bottom of the tree. While it does have the highest equal mean and produced the best final result, it did not consistently perform well.

The representation that had only line nodes fared the worst. This was not anticipated given that all the nodes were contributing to the image, and this had the potential to result in smaller and more efficient trees.

There is, however, a common thread in both of these representations. In both cases there are Line nodes that have exactly one child which must be a terminal. These Line nodes are effectively the leaves of the tree, but they are still counted as functions from the point of view of the GP algorithm. It is therefore possible that when the GP algorithm is selecting nodes to place in the tree, it may select one of these Line nodes as a function node, expecting to be able to create a subtree below it. This, however, is impossible, and will effectively end that branch of the tree. Extending GP with the ability to identify nodes that force finite subtrees into the GP algorithm may improve the performance of the sequence, and lines-only versions of the algorithm.

Creating tree structures to represent these program trees is not the only approach. In many ways the lines representation seems suited to a slightly different approach, where the program is not structured as a tree but rather as a variable length sequence of instructions. This representation would be good if mutation and crossover were modified to work on random subsections of the sequence as opposed to subtrees, otherwise they would only be able to change the nodes at the end of the sequence. While this would have been interesting to test, this is outside of the scope of tree based GP as used in this project, and modifications to GP are outside the scope of this project.

5.2 Comparing the Four Representations

The shapes representation produced the best results. Judging purely by mean fitness, the shapes representation was the best, and the lines was the worst. However, as a human assessing the images in Figure 4.2 as in the human-in-the-loop approach, the mathematical representations seem to have been a lot worse than both lines and shapes. While space and practicality requirements prevent the inclusion of all generated images, considering the other images yields the same result. None of the images drawn using the mathematical representations look like the images they are trying to copy, while the shapes and lines representations do. This is a problem with the fitness function to be discussed later in this chapter. Despite this, the shapes representation had the highest mean fitness and produced recognisable images. This agrees with our original hypothesis that the shapes representation would do well.

The mathematical primitive sets are much larger than the shapes and lines sets. This results in a much larger search space that is much harder to search. Moreover, this primitive set is very small compared to the primitive set used by Sims [25], which many subsequent primitive sets are based on. As such, while their sets are more expressive, it is also significantly harder to find a solution, and as such if there was potential in the mathematical representations, the smaller set may be more likely to find it in the limited time.

It is worth noting that three of the four representations had their best image as The Voyage of Life. With this in mind the means of each image were calculated for the lines and shapes representations (the mathematical ones were ignored as they did not look like the target image in any of the cases). Despite the low number of tests for each image, there was virtually no difference in the means. As such while it is likely that some images will be easier to copy than others (such as a giant white square), that does not appear to have been the case for the images used here.

5.3 Limitations of the Fitness Function

It can be seen from looking at a sample of the images produced by the system used here, as in Figure 4.2, that having a higher fitness value does not necessarily mean that an image looks closer to the original. The images generated with mathematical representations are a good example of this. While they have a higher mean fitness than the lines representation, they look less like the target image than similar images generated with the line representation. This is because the fitness function only takes into account the colour difference of each pixel independent of any other factors such as location, or the colour of neighbouring nodes. This is not, however, how humans view images. Humans see regions of similar colours as "objects". The lack of such "objects" in an image will make it unrecognisable. As the fitness function does not consider regions of colour, it does not factor the appearance of "objects" in its calculation. As such it will classify images that have the approximate colour scheme of the image all over with high fitness, even if the image is not recognisable as the original to a human.

Creating a fitness function that considers regions of colour is more complex than the current implementation, and was originally thought to be unnecessary. If an image were to have a sufficiently high fitness using the implemented fitness function, the produced image would have to be similar, as the only way to have all the colours the same is to be the same image. However, none of the images were near the target fitness and as such had this problem.

5.4 Applicability

The primitive sets tested in this work are not exact copies of all the primitive sets used in other works. Works that use shape-based representations may structure their trees differently. Similarly, works that use mathematical based representations may not use the exact same set of mathematical functions, and may also use different rules to turn real numbers into colours. As such, the findings in this work can not be generalised to all primitive sets.

Even so, these findings have some general applicability. These results are not sufficient to demonstrate that any given primitive set used in other work is not capable of producing arbitrary images. However, these results show that primitive sets that may be thought reasonable, may be less expressive than expected. There are many different primitive sets out there, and while it cannot be said that they are not good enough, it can be said that they should at least be tested to see how good they really are. The performance of the shapes primitive set used here can also be used as an indicator for how a general shapes primitive set can be anticipated to behave, even without being able to provide exact details.

5.5 Image Choice

This project only looked at a small range of possible images. Two of the images were paintings from the last 200 years, and the other was a photo of a statue. This is not a general sample of all possible images. To more fully explore the expressiveness of primitive sets, a wider variety of images should be used. Unfortunately, this was not possible in the time allotted.

While the image used in this project are among the more complicated, they are also among the more interesting. While solving simple geometric images, maybe work better, there is comparatively little to be gained. Given that we have a shapes primitive set, copying a simple geometric image could be trivially solved by a very small tree. Moreover, trying to copy simpler images would not allow for the analysis for more complex images, due to time restraints.

Chapter 6

Conclusions

This project has successfully defined four primitive sets that are capable of generating images based on a fitness function. Irrespective of the particular target image, these primitive sets can generate images, as long as they are used with their expressive ability in mind. As such, any of the four can be used in GP for creating images.

Of these four primitive sets, the shapes representation has the best performance on copying images. The lines representation has the worst performance, however, based on human fitness evaluation, the two mathematical representations seem like they should be ranked lower than the lines. Future work with a modified fitness function may be able to address this issue.

Overall, the resulting images produced in this work are not of very high quality. They are not faithful copies of the original image, but rather stylized versions missing most of the recognisable features and objects. While this means that none of the primitive sets explored should be used widely for all forms of evolutionary art research, they may be suitable for certain tasks.

The definition of a primitive set that is both expressive enough to draw all images, and one that works well with GP is non-trivial. All four primitive sets defined here failed, despite the shapes representation seeming intuitively like it should work. While there is a lot of work that is trying to define a good fitness function, a fitness function is limited by the primitive set that is used to define images. It is therefore important for more work to be done on the expressiveness of primitive sets, and the definition of standard sets for use in GP for evolutionary art.

6.1 Future Work

There are several avenues to pursue from this work. The first is that since this work has not managed to successfully find a good primitive set, further work needs to be done on this. There are several clear alternatives to certain design choices made in this work:

- HSL or HSV can be used as the colour representation instead of RGB
- Having a shape's co-ordinates be a terminal, as opposed to being stored in the shape
- Representing locations using non Cartesian co-ordinates
- Using Linear GP or a linear representation within tree based GP
- Have location as a function of where the node is in the tree, as opposed to a property of the node being drawn

- Using vector representations of images

As well as changes to the representation, work should be done on the fitness function. At the moment it is prone to classifying images that are simply an average of the colours in the image as a good image. Work should be done on a more “intelligent” fitness function. Possibilities include a fitness function that detects whether “objects” exit in the image, or changing the shape of the fitness function to bring lower fitnesses closer together and increase the distance between higher fitnesses. This can be done by exponentiating the current fitness value to an exponent proportional to the number of generations, which will serve the additional purpose of only moving the lower fitnesses together after a certain time.

More work needs to be done on validating the primitive sets that already exist. Testing each of the sets currently in use, on a large set of images of the sort that are desired, with experimental parameters comparable to normal operation. This will help provide results that give information about particular systems and their capabilities.

Work can also be done to explore the effect of adding the knowledge that some GP function nodes may actually act as terminals. It may also be worth trying to copy simpler images. As the images chosen here attempt to sample images from the real world, this may bring with it additional complexity.

Bibliography

[1] ALSING, R. Genetic programming: Evolution of Mona Lisa, December 2008. `http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/` (Accessed 08/10/10).

[2] ATKINS, D. L., KLAPAUKH, R., BROWNE, W. N., AND ZHANG, M. Evolution of aesthetically pleasing images without human-in-the-loop. In *IEEE Congress on Evolutionary Computation* (2010), IEEE, pp. 1–8.

[3] BALUJA, S., POMERLEAU, D., AND JOCHEM, T. Towards automated artificial evolution for computer-generated images. *Connection Science 6*, 1 (1994), pp. 325–354.

[4] BARILE, P., CIESIELSKI, V., BERRY, M., AND TRIST, K. Animated drawings rendered by genetic programming. In *GECCO* (2009), F. Rothlauf, Ed., ACM, pp. 939–946.

[5] BARILE, P., CIESIELSKI, V., AND TRIST, K. Non-photorealistic rendering using genetic programming. In *SEAL* (2008), X. Li, M. Kirley, M. Zhang, D. G. Green, V. Ciesielski, H. A. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K. C. Tan, J. Branke, and Y. Shi, Eds., vol. 5361 of *Lecture Notes in Computer Science*, Springer, pp. 299–308.

[6] CHIO, C. D., BRABAZON, A., CARO, G. A. D., EBNER, M., FAROOQ, M., FINK, A., GRAHL, J., GREENFIELD, G., MACHADO, P., O'NEILL, M., TARANTINO, E., AND URQUHART, N., Eds. *Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoMUSART, and EvoTRANSLOG, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part II* (2010), vol. 6025 of *Lecture Notes in Computer Science*, Springer.

[7] COOK, T. E. Gauguin: generating art using genetic algorithms and user input naturally. In *GECCO (Companion)* (2007), D. Thierens, Ed., ACM, pp. 2647–2650.

[8] DAWKINS, R. *The blind watchmaker*, 1st american ed. Norton, New York, 1986.

[9] DEN HEIJER, E., AND EIBEN, A. E. Comparing aesthetic measures for evolutionary art. In Chio et al. [6], pp. 311–320.

[10] DICKIE, G. Defining art. *American Philosophical Quarterly 6*, 3 (1969), pp. 253–256.

[11] DORIN, A. Aesthetic fitness and artificial evolution for the selection of imagery from the mythical infinite library. In *ECAL* (2001), J. Kelemen and P. Sosík, Eds., vol. 2159 of *Lecture Notes in Computer Science*, Springer, pp. 659–668.

[12] DRAVES, S. The electric sheep screen-saver: A case study in aesthetic evolution. In *EvoWorkshops* (2005), F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, and G. Squillero, Eds., vol. 3449 of *Lecture Notes in Computer Science*, Springer, pp. 458–467.

[13] ECML. RMITGP, November 2009. `http://goanna.cs.rmit.edu.au/~vc/rmitgp/`.

[14] ENGELBRECHT, A. P. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006.

[15] GREENFIELD, G. R. Computational aesthetics as a tool for creativity. In *C&C '05: Proceedings of the 5th conference on Creativity & cognition* (New York, NY, USA, 2005), ACM, pp. 232–235.

[16] JASKOWSKI, W., KRAWIEC, K., AND WIELOCH, B. Learning and recognition of hand-drawn shapes using generative genetic programming. In *EvoWorkshops* (2007), M. Giacobini, A. Brabazon, S. Cagnoni, G. D. Caro, R. Drechsler, M. Farooq, A. Fink, E. Lutton, P. Machado, S. Minner, M. O'Neill, J. Romero, F. Rothlauf, G. Squillero, H. Takagi, S. Uyar, and S. Yang, Eds., vol. 4448 of *Lecture Notes in Computer Science*, Springer, pp. 281–290.

[17] KOZA, J. R. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, USA, 1992.

[18] LI, Y., AND HU, C.-J. Aesthetic learning in an interactive evolutionary art system. In Chio et al. [6], pp. 301–310.

[19] MACHADO, P., AND CARDOSO, A. Computing aethetics. In *SBIA* (1998), F. M. de Oliveira, Ed., vol. 1515 of *Lecture Notes in Computer Science*, Springer, pp. 219–228.

[20] MACHADO, P., AND CARDOSO, A. All the truth about nevar. *Applied Intelligence 16*, 2 (2002), pp. 101–118.

[21] NEITZ, J., CARROLL, J., AND NEITZ, M. Color vision: Almost reason enough for having eyes. *Opt. Photon. News 12*, 1 (2001), pp. 26–33.

[22] OH, J. C., AND ZAJEC, E. Evolving artistic styles through visual dialogues. In Chio et al. [6], pp. 261–270.

[23] ROSS, B. J., RALPH, W., AND ZONG, H. Evolutionary image synthesis using a model of aesthetics. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation* (Vancouver, 6-21 July 2006), G. G. Yen, L. Wang, P. Bonissone, and S. M. Lucas, Eds., IEEE Press, pp. 3832–3839.

[24] SAH, S. B. M., CIESIELSKI, V., AND D'SOUZA, D. J. Refinement techniques for animated evolutionary photomosaics using limited tile collections. In Chio et al. [6], pp. 281–290.

[25] SIMS, K. Artificial evolution for computer graphics. In *SIGGRAPH* (1991), J. J. Thomas, Ed., ACM, pp. 319–328.

[26] STOUT, C. J. Multicultural reasoning and the appreciation of art. *Studies in Art Education 38*, 2 (1997), pp. 96–111.

[27] SVANGÅRD, N., AND NORDIN, P. Automated aesthetic selection of evolutionary art by distance based classification of genomes and phenomes using the universal similarity metric. In *EvoWorkshops* (2004), G. R. Raidl, S. Cagnoni, J. Branke, D. Corne, R. Drechsler, Y. Jin, C. G. Johnson, P. Machado, E. Marchiori, F. Rothlauf, G. D. Smith, and G. Squillero, Eds., vol. 3005 of *Lecture Notes in Computer Science*, Springer, pp. 447–456.

[28] TAKAGI, H. Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. *Proceedings of the IEEE 89*, 9 (sep 2001), pp. 1275–1296.

[29] XU, Q., D'SOUZA, D. J., AND CIESIELSKI, V. Evolving images for entertainment. In *IE* (2007), L. Hjorth and E. Milne, Eds., vol. 305 of *ACM International Conference Proceeding Series*, ACM, p. 26.

[30] YUAN, J., AND GONG, D. Large population size IGAs with individuals' fitness not assigned by user. In *ICIC (2)* (2008), D.-S. Huang, D. C. W. II, D. S. Levine, and K.-H. Jo, Eds., vol. 5227 of *Lecture Notes in Computer Science*, Springer, pp. 267–274.