Evolutionary Art with Cartesian Genetic Programming

Laurence Ashmore[1], and Julian Francis Miller[2]

[1] Department of Informatics, University of Sussex, Falmer, BN1 9QH, UK
emoai@hotmail.com
http://www.gaga.demon.co.uk/
[2] Department of Electronics, University of York, Heslington, York, YO10 5DD, UK
jfm@ohm.york.ac.uk
http://www.elec.york.ac.uk/staff/jfmhome.htm

Abstract. Techniques from the field of Evolutionary Computation are used to evolve a wide variety of aesthetically pleasing images using Cartesian Genetic Programming (CGP). The challenges that arise from employing a fitness function based on aesthetics, and the benefits that CGP can provide, are investigated and discussed. A significant piece of software was developed that places a focus on providing the user with efficient control over the evolutionary process. Several 'non-user' fitness functions that assess the phenotypes and genotypes of the chromosomes were also employed with varying success. To improve these results, methods of maintaining diversity within the population that take advantage of the neutrality of CGP are implemented and tested.

1 Introduction

Evolutionary art uses evolutionary computation to evolve images or aesthetically pleasing structures. The kinds of images that can be evolved greatly vary and heavily rely on the particular representation being used. The main difference between evolutionary art and other search problems is that the fitness of an image is based on something that is very hard to describe or maybe even to understand. The attractiveness of an image is also a personal thing that will differ between people. This makes evolutionary art a very interesting topic in terms of evolutionary search. A lot of search problems are concerned with optimisation and convergence towards a solution. With evolutionary art the search is more exploratory, with divergence and diversity being the key factors. There can never be a strict fitness function when assessing the attractiveness of an image so it is the user who provides the fitness of an image in the majority of evolutionary art. Nearly all evolutionary art programs have the same interface; a number of pictures or shapes are displayed on the screen within a grid so that the user can view the whole population at the same time. The user selects a number of their favourite images that are then used to create the next population. There are many different ways in which people have genetically represented images and the way in which evolution is simulated also varies.

In this paper we use a relatively new form of Genetic Programming [1] called Cartesian Genetic Programming (CGP) [3]. This is the first use of this technique in evolutionary art. There are several features of CGP that make it very suitable for using in evolutionary art. These features include the way in which program outputs are functionally related and the neutrality that is present in CGP.

The plan of the paper is as follows. In section 2 we review a few evolutionary art systems that are close to that presented here. In section 3 the Cartesian Genetic programming method is explained. Section 4 describes the implementation. In section 5 some evolved images are presented. We end with some brief conclusions.

2 Related Evolutionary Art Systems

Karl Sims was the first person to use genetic programming to evolve 2D art and inspired most of the currently available evolutionary art programs [5]. His most famous work was an art installation entitled "Genetic Images" that displayed 16 images for the public to pick their favourites from. The most aesthetically pleasing images survived and were mutated to create the next generation. The images created from this set-up greatly vary in appearance and are curiously interesting. Each image is represented by a single LISP expression tree that outputs the colour of a pixel dependant on its coordinates. Sims used a very rich function set, containing image-processing functions (blur, convolution and gradient functions) as well as simple mathematical functions. The colour in an image is introduced by functions such as color-noise and warped-color-noise being present in the tree.

Steven Rooke has created some of the more impressive evolutionary art images, mainly due to his inclusion of evolvable fractals in his function set [4]. The images are represented as LISP parse-trees and are evolved using crossover and mutation on the trees. The output from the program is interpreted by a colour mapping function that maps the scalar values to RGB vectors. A typical population contains 150 individuals, these are initially composed of seeded (previously good genotypes) and random individuals. Each individual in the population is given a score, the better the score the higher the probability of it going into the next generation and having offspring. There is no limit to the size of the representation (other than computer memory) so highly complex trees can be evolved.

Penousal Machado has developed an evolutionary art program called NEvAr (Neural Evolutionary Art) [2]. This uses GP with a set of very simple functions. The idea behind this is that using complex functions incorporates a bias into the search and makes it more confined. Using only simple functions opens out the search space, unfortunately it also increases the time taken to breed 'good' images. This is in contrast to Sims' genetic art that used a very rich function set.

3 Cartesian Genetic Programming

Cartesian genetic programming is a form of genetic programming where the program is represented by a directed graph of indexed nodes [3]. The graph has a set of n_i inputs that are indexed as nodes 0 to n_i-1, a set of n_n nodes and a set of n_o outputs that are taken from the last n_o nodes. Each node has a number of inputs and a function that gives an output based on the inputs. The genotype is a list of integers that determine the connectivity and functionality of the nodes. These can be mutated and crossed over to create new directed graphs.

The genotype is of fixed length however the graph described by it is not. This is due to there being expressed and unexpressed nodes in the genotype. The genotype may contain nodes that are not connected to the output nodes so are not expressed in the phenotype, this is called node redundancy. As well as node redundancy there is also functional redundancy and input redundancy. Functional redundancy is where a set of nodes implement a more complex function that could be implemented with fewer nodes. Input redundancy is where some of the node inputs are not used by the function of the node. This redundancy provides CGP with greater neutrality [3][6] when compared with standard GP, and hence a very different behaviour. The presence of a genotype-phenotype mapping allows different genotypes to map to the same phenotype, creating many plateaus in the search space. When a plateau is reached genetic drift may occur across the plateau. Genetic drift is the changing of unexpressed genes, or nodes, in the genotype that may lead to a later improvement in fitness when they are expressed. With a given mutation rate it may not be possible to acquire an offspring with an improved fitness. However, if genetic drift occurs then a later offspring may have the ability to create a fitter individual, enabling escape from local minima.

In standard GP the evolved program only has one output (although the output could be a vector value), in CGP it is possible to have as many outputs as necessary. These output values can share nodes so can functionally relate to each other much closer than outputs from several separate GP trees.

4 Implementation

4.1 Representation

In evolutionary algorithms the chromosome representation is a key factor. It has a large influence on the success of the algorithm and can dictate the search space and traversal through it. Different representations apply themselves better to certain problems, choosing the wrong representation for a given problem can often lead to bad results. The graph has two inputs, the x and y coordinates of a pixel in the image, and three outputs, the three colour channels (red, green and blue) for that pixel. The program represented by the chromosome maps each coordinate, based on its value, to a specific colour, specified by the RGB values. Hence changing the functions and connectivity of the nodes will change the colour values of each pixel, and so change the image. The genotype is stored as an integer array with length (n*4)+3 where n is the number of nodes. The last three integers in the chromosome are the output pointers for the red, green and blue colour channels.

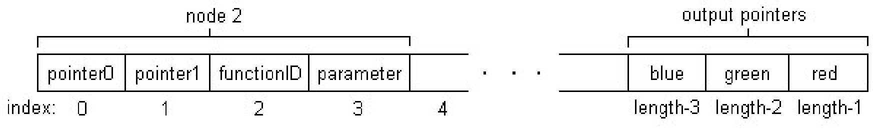

Fig. 1. Schematic of genotype

The first node is referenced as node2 so the two inputs for the x and y coordinates can be indexed as 0 and 1. Since each pixel position has the same equation defining its colour mapping there will be a relationship between pixels which should make for interesting images. The number of nodes in the chromosome is related to the complexity of an image. The greater the number of nodes then the more functions that can be used in the equation defining the image. Obviously, increasing the number of nodes greatly increases the search space and also noise within the search space. Although large chromosomes will be able to produce more complicated and interesting images but the user may not want to spend much time evolving chromosomes, it is important to be able to control this.

The outputs were primarily chosen to represent red, green and blue colour channels. A lot of previous evolutionary art programs have had a single output that is mapped to colours using a colour lookup table. Using this method requires a preset or separately determined colour-mapping table (cf. Rooke). It is much simpler to have three outputs, one for each colour channel, so that the colour is completely defined by the program output, rather than the output and its colour mapping. RGB is a standard colour model and very intuitive because it uses the primary colours to pinpoint the pixel values in the colour space. By using the RGB colour representation each channel has the same effect on the image (e.g. each value describes the brightness of a colour). With HSB (hue, saturation, brightness) each channel has a very different affect on the final image. By switching around the pointers very different images are created. With RGB the different outputs can work together to produce a very interesting image, in other words the whole is greater than the sum of its parts. With a HSB mapping the outputs work against each other, they describe very different characteristics of the image, removing the benefit of functionally related outputs.

In CGP each node normally defines the inputs to the node and the function only. The functions are limited by having outputs between 0 and 255. To increase the flexibility a further parameter has been added to each node which may or may not be used by the specified function. The number of inputs to each node is determined by which functions are allowed. It was decided to have two inputs to each node (although both inputs need not be used by every function). Hence, each node consists of four integers.

One option was to have the last three nodes in the chromosome taken as the outputs, as in standard CGP, but this would create several problems. This can be illustrated by trying to evolve a black and white image; the last three nodes would have to have all the same pointers, functions and parameters. Instead there will be three separate pointers that can point to any node and take that value as the output. This makes it easier to have the outputs the same value. Many of the good pictures found early on in the development had very similar colour channels, with maybe only one or two nodes affecting one of the channel outputs after the other outputs had been taken. This would be a lot harder to control if each output had an extra function affecting it. Another reason for having the output pointers separate is that from an early stage it was obvious that control over colour is a very important factor. By having the pointers not linked to a function the colour channels can be rotated without swapping any important node information.

4.2 Functions

Choosing the function set was very time consuming. It is important that there are enough functions to be able to create complex images but not so many that there are functions that have a negative effect on image fitness. Such functions make traversal through the search space very laborious. Picking functions is difficult as it is very hard to determine what effect any given function will have on an image. Early tests using chromosomes with one node or two nodes gave indications on the utility of certain functions. Many functions that were tested and omitted from the final set because they either had no noticeable effect or created too much "noise" in the search space.

0: input1 | input2;
1: parameter & input1;
2: (input1 / (1.0 + input2 + parameter));
3: (input1 * input2) % 255;
4: (input1 + input2) % 255;
5: if(input1>input2) input1 - input2; else input2 - input1;
6: 255-input1;
7: abs(cos(input1)*255);
8: abs(tan(((input1%45)*π)/180.0)*255));
9: abs(tan(input1)*255)%255);
10: sqrt((input1-parameter)^2 + (input2-parameter)^2); (thresholded at 255)
11: input1%(parameter+1)+(255-parameter);
12: (input1 + input2)/2;
13: if (input1>input2) 255*((input2+1)/(input1+1));else 255*((input1+1)/(input2+1));
14: abs(sqrt(input1^2-parameter^2+ input2^2-parameter^2)%255);

Fig. 2. Empirically chosen function set

4.3 Population

The population size is very important because all of the individuals need to be displayed on the screen at the same time for fair evaluation. With a large population the images will need to be shrunk and may lose important detail. With a small population there may not be enough variation in the images and the user will need to keep generating new populations with the same parents over and over again.. Since random populations are relatively boring, the program initiates with a population constructed from previously generated good chromosomes. These are loaded at random from a 'pool' of good genotypes and provides the user with an interesting collection of chromosomes to begin evolution with.

4.4 Search Operators (crossover and mutation)

Applying search operators to graphs is not trivial so care has to be taken in the design of mutation and crossover. Deciding which point to mutate is done by picking a random point along the chromosome, the mutation rate specifies how many points are randomly picked. When a point is picked it needs to be determined what the point represents, whether it is a pointer, function or a parameter. If it is a pointer then it can be mutated to a random integer between 0 and the number of the node before the node being mutated. The function value is limited by the number of functions. The value that the parameter can be is determined by the node function. Some functions use a value from 0 to 255, others 0 to 50 or the parameter may be unused. Specific colour channels can be locked to increase control over variation. To do this there needs to be a record kept of which nodes are used by each output. If a colour channel is selected to be locked then all the nodes that define the equation for that colour channel must remain static. When picking a point for mutation it can be made sure that the point is not part of any locked node. If all of the colour channels are locked then only the redundant nodes will be mutated and the output will not change.

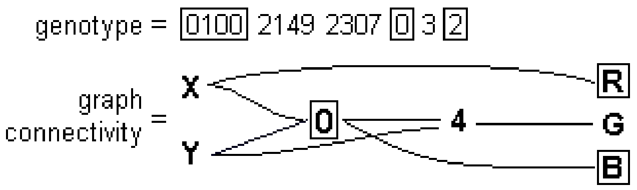

Fig. 3. The effect of locking the red and blue channels (locked points shown boxed).

Fig. 3 shows which points are locked when the red and blue colour channels are locked. The red and blue output pointers are locked and all the points in the first node are locked because the blue channel takes its output from it. It was found to be often useful to only mutate the parameter part of the nodes, leaving the graph connectivity and functionality unaltered. This can be easily achieved by making sure that the point picked for mutation is always a parameter.

Crossover is not normally used when evolving graphs because it tends to be far too destructive. Since evolutionary art does not have a well-defined fitness function and the whole process is a lot more exploratory, having operators that disturb the genotype is more of a good thing than a bad. Early tests also showed that the crossover could result in an image with the 'form' of one image and the colour from another, as can be seen in Fig. 4. This is a very strong tool for achieving a desired image. It was found that nodes near the beginning of a chromosome code for the image layout while nodes towards the end code for the colour changes. It would be interesting to see whether representations other than CGP also have this property.

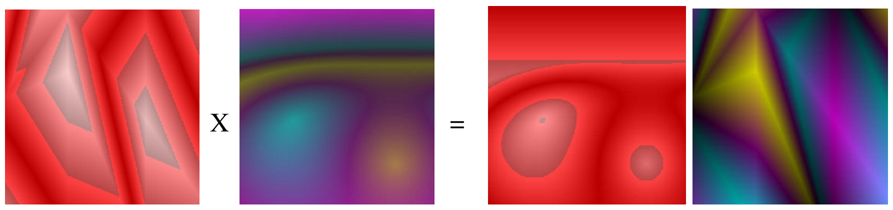

Fig. 4. Crossover of parents (left) to produce two children with inherited features.

Crossover points are picked between whole nodes so information within the nodes is retained. In the offspring, the nodes before this point will all come from one parent and the nodes after that point, including the output pointers, will come from the other parent. Genotypes of different lengths can be crossed over by making the offspring have the same number of nodes as the largest parent. The offspring could be forced to have the length of the shortest genotype but this makes crossover more restrictive. The crossover point is picked from the shortest genotype and nodes before this point are copied to the offspring. The nodes after this point will have to come from the longest genotype.

The direction of search and the fitness of individuals is specified by the user so the user needs complete control of the mutation rate and whether crossover occurs or not. During autonomous evolution (see Section 4.5) the mutation rate is controlled by using adaptive mutation. The mutation rate is quite high during the first few generations and decreases as evolution progresses. This is to avoid early convergence by keeping the jumps through the search space quite large and to enable small refinements later on.

4.5 Autonomous Evolution

The major time consuming factor in evolutionary art is the fitness evaluation, as it requires human intervention. Having the ability to 'autonomously' evolve images can greatly increase the time taken to find attractive images. In terms of aesthetics any 'non-user' fitness function is going to be naïve, however they can still be useful. A randomly seeded population will nearly always appear uninteresting making it very hard for the user to know which direction to take the evolution. Using a predefined fitness function, such as the evolving towards greater complexity (more used nodes in the genotype) or for circular objects in the image (results from a Hough Transform operation), often result in a population that is easier for the user to work with.

The top three images in each generation are used to seed the next, and an elitism replacement strategy is employed. Diversity is maintained by forcing the three images to have a certain degree of difference (calculated pixel by pixel). Any parent genotype that survives two consecutive generations is replaced with a different genotype with an identical phenotype (if one is present in the population). This increases genetic drift in the population and achieves improved results. This is shown most convincingly when the fitness function is the Euclidean distance from a user chosen Jpeg image. Fig. 5. shows the fitness of the top individual for 4000 generations when using and dismissing the above technique.

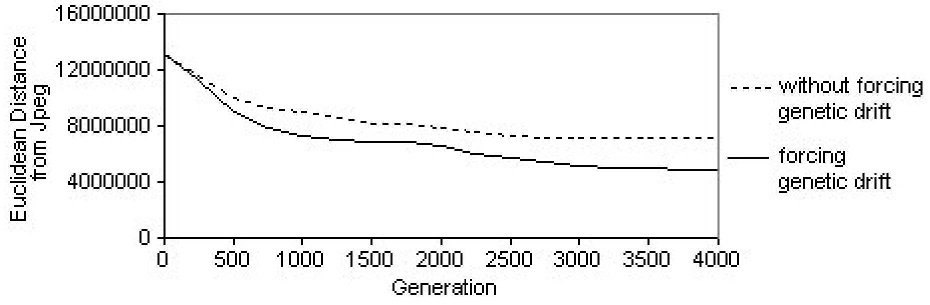

Fig. 5. Graph demonstrating the benefits of genetic drift when using a 'non-user' fitness function (results have been averaged over 20 evolutionary runs).

5 Results

In Fig. 6 are shown some evolved pictures from separate populations using chromosomes with 20 nodes and were all evolved for about 10 generations from a random genotype. They give a good indication of the range of pictures that may be produced in a very short space of time. Fig. 7 shows some images evolved for 500 generations without human intervention by choosing a fitness function that is equal to the number of nodes used in the phenotype with a bias towards various functions being used in the chromosome. When evolving pictures there are often images that seem like they 'carry on' outside the boundary of the image. It is interesting to be able to see what shapes and patterns occur around the images and to see the whole effect of the algorithm that determines the picture. This can be achieved by allowing the user to enter the start and end co-ordinates that are given to the chromosome when creating the image. Invalid colour values are cropped to either 0 or 255. This adds to the program by giving the user a means to further explore the images they create.

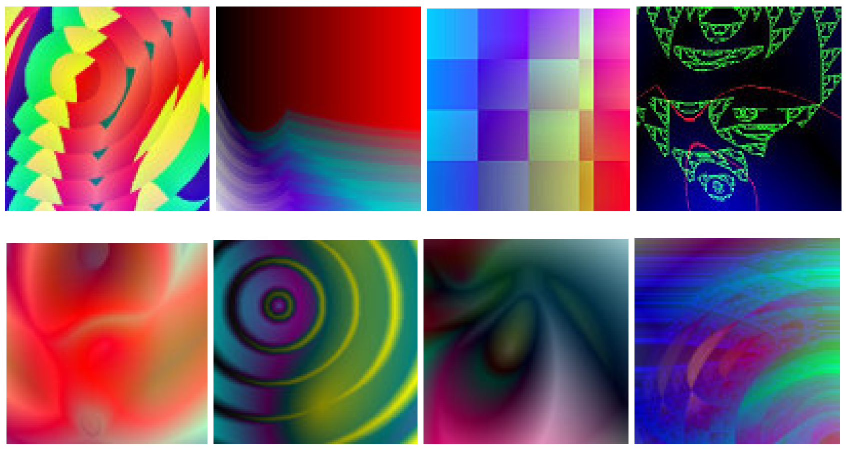
Fig. 6. Evolved images with 20 nodes from separate populations

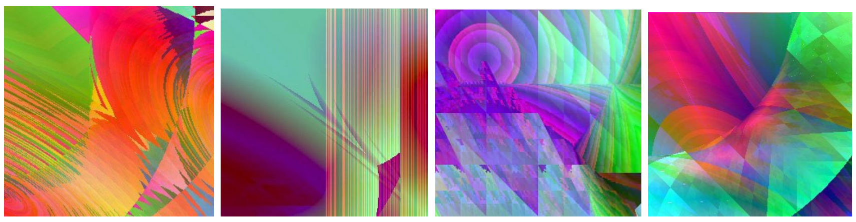
Fig. 7. Automatically evolved images with 50 nodes

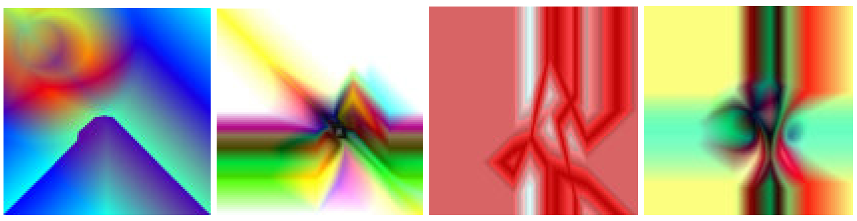
Fig. 8. Some evolved images extended to 512x512 pixels

Interesting things can happen just out of view on the normal image or the image may show a small part of a larger and more interesting ‘structure’. Some examples of this are shown in Fig. 8. Since some of the functions in the function set carry numerical parameters, it is possible to iterate through different parameter values after a satisfactory image has been obtained. In most cases small changes to parameters result in subtly different images. These images can be animated. Some of these were found to be very effective and gave the appearance at times of flying through the image. The ability to animate chromosomes was initially added to the program as a way to fine-tune an image. Since the evolutionary process is stochastic it can sometimes be frustrating if a chromosome continually fails to evolve to anything more interesting.

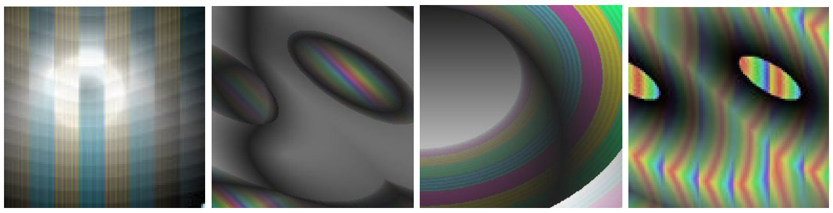

Fig. 9. Some evolved images using HSB mapping

By animating the chromosome it is possible to visualise what the possible effects of mutating specific parameters can be. The user can select the desired frame and then place that chromosome back into the population to continue evolution with. This is a powerful tool because it enables manual manipulation of the chromosomes. It also allows the user see what possible directions the evolution could be taken and gives the user a greater understanding of how the images are created.

6 Conclusion

We have described an evolutionary art system that uses a graph based form of Genetic Programming called Cartesian Genetic Programming. The method has a number of advantages in that it allows a multi-output program that encodes for RGB or HSB. Particular combinations of outputs can be locked by the user and the remaining outputs evolved. The method also uses parameters as the arguments to some primitive functions that can be iterated to create animated art, this can be aesthetically pleasing but also can assist the designer by showing them regions of parameter space that have interesting visual effects. The method also benefits from the highly neutral search possibilities that the Cartesian encoding allows.

References

1. Koza, J. R. : Genetic Programming, MIT Press, London, (1993)
2. Machado, P. and Cardoso, A. : NEvAr --- The Assessment of an Evolutionary Art Tool. In: Wiggins, G. (Ed.). Proceedings of the AISB00 Symposium on Creative & Cultural Aspects and Applications of AI & Cognitive Science, Birmingham, UK, 2000
3. Miller, J. F. Thomson, P. : Cartesian Genetic Programming, Proceedings of the 3rd European Conference on Genetic Programming, Edinburgh, Springer, Berlin (2000) 121-132
4. Rooke, S. : Eons of Genetically Evolved Algorithmic Images. In: Bentley P. J. and Corne D. (eds.): Creative Evolutionary systems, Morgan Kaufmann, San Francisco (2002)
5. Sims, K. : Artificial Evolution for Computer Graphics. Computer Graphics, Vol. 25, (1991) 319-328
6. Yu, T. and Miller, J. F. : Neutrality and the evolvability of Boolean function landscape. Proceedings of the Fourth European Conference on Genetic Programming, Springer-Verlag, Berlin (2001) 204-217