Giving Colour to Images

Penousal Machado[1,2]; André Dias[2]; Nuno Duarte[2]; Amílcar Cardoso[2]
[1] Instituto Superior de Engenharia de Coimbra; Quinta da Nora, 3030 Coimbra, Portugal;
[2] CISUC – Center for Informatics and Systems, Univ. Coimbra, 3030 Coimbra, Portugal;
machado@dei.uc.pt; adias@student.dei.uc.pt; nduarte@student.dei.uc.pt; amilcar@dei.uc.pt;

Abstract

This paper is about the colouring of greyscale images. More specifically, we address the problem of learning to colour greyscale images from a set of examples of true colour ones. We employ Genetic Programming to evolve computer programs that take as input the Lightness channel of the training images and output the Hue channel. The best programs evolved can then be used to give colour to greyscale images. Due to the computational complexity of the learning task, we use a genome compiler system, GenCo, specially suited to image processing tasks.

1 Introduction

The work presented here is part of a wider research project, NEvAr, whose aim is to build a constructed artist (i.e. a program that generates artworks autonomously).

NEvAr is an Evolutionary Art Tool inspired on the work of K. Sims (1991). It relies on Genetic Programming (GP) to evolve populations of images, based on aesthetic principles. Fitness assignment plays, like in most Evolutionary Computation systems, a key role since it guides the evolutionary process.

NEvAr can be used as an Interactive Evolution tool. In this mode of execution the user supplies the fitness values to the evolved images. It can also be used as a fully autonomous system. In this case fitness is assigned through an explicit fitness function, which takes into consideration several complexity estimates of the images (Machado, 2002).

However, the fitness assignment procedure only takes into account the lightness information of the images, discarding the hue and saturation information. Therefore, in this mode of execution, we are limited to greyscale images. A full description of NEvAr and of the automatic fitness assignment can be found in (Machado, 2002).

There are good theoretical and artistic reasons to deem colour less important than lightness. The development of the colouring procedures of systems like AARON (Cohen, 1995) is, to some extent, based on this notion. However, this collides, at least apparently, with the importance given to colour by some of the most prominent painters (e.g. (Kandinsky, 1991)). Moreover, NEvAr's limitation to greyscale images, in its autonomous version, was frustrating, to say the least. In this paper we address the problem of giving colour to greyscale images.

An analysis of the role of colour and the way colour is assigned, particularly in abstract art, leads to the conclusion that artists (certainly not all, but at least a significant proportion) usually work with a limited colour palette, and that the spatial relation between colours usually follows some rules. This is consistent with the view that each artist constructs its own artistic language, which complies with an implicit grammar. It is also consistent with the approach used in AARON to colour its drawings (Cohen, 1995; Cohen, 1999).

The idea of creating a program to give colour to the greyscale images created by NEvAr emerged naturally. Unfortunately this poses several problems. Our system is based on a non symbolic approach and produces bitmap images, hence there is no clear definition of closed forms, shapes, etc.

Therefore, although we could define a palette to work with, assigning the colours of that palette to specific forms would be difficult since we have no forms to begin with. Even assuming that the forms could be properly identified by some sort of pre-processing method, assigning the right colour to each shape and keeping a proper spatial relation among colours would still be a problem, due to the unstructured nature of the output.

Additionally, the creation of a colouring system, by itself, doesn't appear to be an easy task, involving the choice of an adequate set of palettes, establishing a consistent colouring grammar, etc.

Taking these facts into consideration, and also the fact that the generality of this type of approach would be limited, we decided to abandon this idea. Instead, we are trying to create a system that learns to colour images from a set of training ones.

This approach has, potentially, several advantages over a built-in colouring procedure, namely: we don't need to code by hand a set of colouring rules; the results of the system are less predictable; we can use paintings made by well-known artists as training set, hence learning to colour images according to their style.

Additionally, it's also an indirect way of testing if the colouring procedures followed by some artists can be formally expressed.

The paper is structured as follows: In the next section we describe our current approach to colouring images; In Section 3, we present some experimental results attained by this approach and make a brief analysis; finally, in section 4, we will draw some conclusions and present our ideas for future research.

2 Our Approach

GP is one of the most recent Evolutionary Computation techniques. Its goal is to evolve populations of computer programs, which improve automatically as evolution progresses (Banzhaf 1998).

Due to the outstanding influence of the work of Koza (1992) it is common, within the Machine Learning community, to associate the term GP to the evolution of tree structures. In this paper we follow this "classical" definition. Therefore, when we talk about GP we are talking about the evolution of tree structures, which are built from a set of functions (f-set) and terminals (t-set). The internal nodes of the tree are members of the f-set, and the leafs are members of the t-set.

In our approach we use GP to evolve populations of programs that give colour to greyscale images. We start by selecting a true-colour training image (or set of images), which is split in its Lightness, Hue and Saturation channels (see Fig. 1).

The evolved programs take the greyscale image corresponding to the Lightness channel as input, and output a greyscale image. The output is compared with the Hue channel of the training image, the closer the output is to the desired one, the higher the fitness. The same procedure can be applied using the Saturation channel, to evolve programs that generate the saturation information.

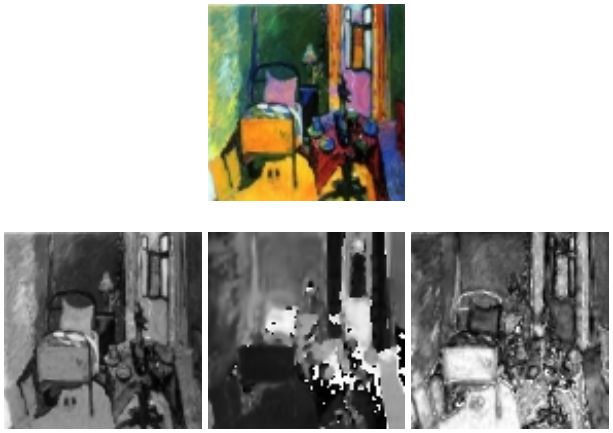

Figure 1: The original image, and the corresponding Lightness, Hue and Saturation channels.

As evolution progresses, the quality of the individuals increases and, eventually, we find programs which generate colourings very close to the original ones.

It is important to notice, however, that this is not enough – the idea is to use these programs to give colour to different images. The fact that a program produces an output that exactly matches the training one does not guarantee that it will produce an interesting colouring on different images. In other words, the evolved programs must generalise well. To promote their generalising capabilities, we took some precautions in the selection of the function set and also in the construction of the fitness function. In the remainder of this section, we describe the options taken and give justification for these options.

2.1 Implementation details

A first word goes to how the output of each program is calculated: given a particular individual, it is run for each of the pixels belonging to the training image (or images). Therefore, assuming a training image of 100*100 pixels, each individual must be run 10000 times. Each execution of an individual implies the transversal of its tree and calling, for each node the corresponding function.

When we take into account that the individuals can easily reach sizes of several thousand nodes, and that GP popu-

lations usually contain several hundred individuals, the conclusion that the execution step is of considerable computational weight clearly follows. In order to minimize this problem we implemented our system with GenCo, a Genome Compiler system specially suited for image processing tasks (Machado, 2001).

A Genome Compiler is a GP system that makes online compilation of the evolved programs. In situations like the one previously described, i.e. in which each individual must be executed several times, this type of system can provide significant speed improvements, since each individual is compiled once and the resulting machine code executed several times (10000 considering a training image of 100*100 pixels). In this scenario, genome compiler systems are, typically, 50 to 100 times faster than standard C based GP implementations (Fukunaga 1998; Nordin 1994; Nordin 1995; Machado 2001).

Next we describe the design options made in the selection of the function and terminal set, and the reasons that justified them.

2.1.1 Function and Terminal Set

Our problem has some similarities with the symbolic regression of functions or images (Nordin 1995; Koza 1992). There are, however, some important differences. In a symbolic regression task, one would usually resort to a function set composed by the arithmetic operations and the *if* statement, and use as terminal set the variables X, Y. This type of set-up unavoidably results in programs whose output depends exclusively on the coordinates of the pixel being calculated.

In an attempt to solve this problem, we added to the terminal set the lightness value of the pixel being evaluated, thus giving more information to the program. This allows the output to vary in accordance to changes on the lightness channel. Additionally, we also added the lightness values of the adjacent pixels, so that the programs have access to the surrounding context of each pixel.

In what concerns the function set, we decided to keep the "traditional" one. The reason for this choice is threefold: it was deemed sufficient for a first approach; the inclusion of more complex functions (e.g. for-next loops, or high level constructs) would severely increase the computational weight of the evaluation step; the inclusion of this type of functions doesn't necessarily benefit the evolutionary process, in fact, the opposite can, and frequently happens (see, e.g., (Banzhaf, 1998)).

Taking all this factors in consideration we used the following function and terminal sets:

- F-set = {+, -, *, %, if}, where % stands for the protected division operator, and if for the *if-less-then-else* statement (Koza, 1992)

- T-set = {X, Y, A..I}, where X, Y are variables corresponding to the coordinates of the pixel, and A..I are the lightness values of the pixel being calculated and of the surrounding ones.

The results achieved were very disappointing, basically because the behaviour of programs continued to be determined, almost exclusively, by the values of the X and Y variables. This is clearly undesirable, since it means that we are not evolving programs that give colour to images according to their lightness channel, we are merely evolving a function that, when applied over an interval of X, Y values, generates the hue channel of the training image. Thus, we are only memorizing the training instance(s).

Taking the X, Y variables from the terminal set resulted in having poor evolution, and convergence to trivial and uninteresting colourings (e.g. having as a result the predominant colour of the training image, or always assigning to a specific lightness value the same hue or saturation).

It was clear that the variables X, Y were not producing any desirable impact on the programs. Therefore, we decided to delete them from the terminal set. It was also clear that without these variables the programs hadn't enough information to determine the appropriate colour for each pixel, since the programs only have a local view of the lightness channel (the pixel being evaluated and the nine surrounding ones). To compensate this lack of information, we introduced a new function, *get(Δx,Δy)*, whose behaviour can be described as follows: assuming that the current pixel has the coordinates a, b, it returns the lightness value of the pixel situated on *(a+Δx, b+Δy)*.

Since the *get* function provides a way to access the lightness values of the surrounding pixels, there was no reason to make these values as part of the terminal set. Accordingly, our function and terminal sets became:

- F-set = {+, -, *, %, if, get}

- T-set = {E}, where E stands for the lightness value of the pixel being evaluated.

This set-up proved to be adequate, allowing the evolution of suitable colouring programs with good generalization capabilities, as will be shown in section 3. In the following section we focus on the used fitness functions, and on the grounds for using them.

2.1.2 Fitness Functions

We start by describing the fitness function used when we are evolving the saturation channel of an image from the lightness one. In this situation we use the root mean square error between the desired output and the real one to assign fitness, i.e. considering that *I* holds the desired saturation values and that *P* is the output of a program the fitness is given by the following formula:

$$s = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max\_x} \sum_{y=1}^{\max\_y} \left(I(x,y) - P(x,y)\right)^2}}{\max\_x \times \max\_y}} \quad \textbf{(1)}$$

When we are trying to evolve the Hue information, this formula must be slightly altered due to the circular nature of the Hue channel. Assuming that the images take values in the [0, 1] interval, the above formula yields a maximum distance when *I(x,y)* = 1 and *P(x,y)* = 0 (or vice-versa). However, in this situation the difference in hue would be quite small and probably unnoticeable to a human observer. Therefore, we use the following formula:

$$h_a = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max\_x} \sum_{y=1}^{\max\_y} \left[\min_\theta \left(I(x,y), P(x,y)\right)\right]^2}}{\max\_x \times \max\_y}} \quad \textbf{(2)}$$

where *I* holds the desired Hue values, *P* the program's output, and min_θ returns the minimum angle distance between the two values. This fitness function can be further improved if we take into consideration that the perceived difference in hue depends on the lightness of the pixel (e.g. if the pixel as lightness equal to zero it will always be black, no matter what hue we assign to it; even when the lightness is only close to zero, giving to that pixel a hue different than the desired one will hardly be noticeable to a human viewer). Taking this into account, and considering that *L* holds the lightness information we obtain the following formula:

$$h_{a'} = \frac{1}{1 + \frac{\sqrt{\sum_{x=1}^{\max\_x} \sum_{y=1}^{\max\_y} \left[\min_\theta \left(I(x,y), P(x,y)\right) \times \left(1 - 2 \times \left|0.5 - L(x,y)\right|\right)\right]^2}}{\max\_x \times \max\_y}} \quad \textbf{(3)}$$

The tests conducted using formulas 2 and 3 as fitness functions yield deceptively good results. In the early steps of evolution fitness increases steadily and swiftly, giving the impression that an adequate colouring program will be easily found. However, after a few hundred generations, the improvements in fitness drop suddenly and evolution seems to halt. This wouldn't be a cause for concern if the evolved programs solved the problem at hand satisfactorily. To some extent they do, since they usually have high fitness values, relatively close to the maximum attainable fitness. The problem is that a qualitative analysis of the results reveals that the colourings are usually uninteresting from an aesthetic perspective – typically, only a small subset of the original colours is used, resulting in the loss of nuances that make the original colouring work. This situation becomes more severe when the training images have a large area filled with a particular hue value, and other areas filled with hue values close to that one.

To better explain the problem we will use an example: consider, for instance, a landscape painting mostly filled with green (for the grass and trees) and with a blue sky (green and blue are relatively close in the colour spectrum); a program that outputs the hue corresponding to the green colour will have a relatively good fitness, since it is not penalised in the dominant green area of the image and only suffers a little penalisation on the blue area. To make things even worse, such a program is easy to evolve, so it will be found in a small number of generations. Changing it by mutation or crossover will tend to decrease its fitness, meaning that the fittest descendents will tend to be similar to it. Therefore, in a few generations the population will be dominated by similar programs, which will begin to increase in their size to protect themselves from destructive crossover and mutation; soon the increase in size becomes exponential and evolution becomes impossible (the exponential growth of program size is usually called bloat problem; a more detailed explanation of why bloat occurs can be found in (Banzhaf, 1997)).

In an attempt to force our system to evolve more interesting colourings, covering a wider colour spectrum, we decided to add another factor to our fitness function. A description of the procedure used to calculate that factor follows.

We start by taking the *I* image (that holds the desired hue values) and decrease its colour depth, using the optimised median cut algorithm, obtaining an image *I'* composed by only 16 different hue values ($v_1 \ldots v_{16}$). Then we count how many pixels exist of each different hue and store the pixel count values in an array, $A_{I'}$. For each pixel of image *P*, which holds the output of the program, we determine the closest hue value, v_i, and the distance Δv between the pixel value and v_i. We add to $A_{P'[i]}$ the value 1-Δv, thus if the pixel's hue matches exactly one of the sixteen hues present in the training image we add one, when it doesn't match exactly we add slightly less. After performing this procedure for all the pixels of the output image we compute the following formula:

$$h_b = \frac{1}{1 + \frac{\sqrt{\sum_{i=0}^{15} \left(A_{I'}[i] - A_{P'}[i] \right)^2}}{16}} \quad \textbf{(4)}$$

Thus, we are basically comparing the number of pixels of each hue value of the output and target image. Returning to our previous example of the green and blue landscape, if the output image has the same amount of blue pixels and green pixels than the original (and also assuming that it is the exact blue and green tone), formula 4 will give a value close to one. However, if the output image is dominated by the green colour there will be a huge discrepancy between the amount of green and blue of the two images, and therefore H_b will be close to zero. Notice that, in what H_b is concerned, the placement of the colours has no real influence on the resulting value. To further force a wider coverage of the colour spectrum, we also used the following modification to this formula:

$$h_{b'} = \frac{1}{1 + \frac{\sqrt{\sum_{i=0}^{15} \left(\frac{A_{I'}[i] - A_{P'}[i]}{\max(A_{I'}[i], A_{P'}[i])} \right)^2}}{16}} \quad \textbf{(5)}$$

which ensures that all sixteen different hues have the same weight in the calculation. Resorting again to our example, and considering that we have a small yellow area (for the sun), when we use formula 5 having the correct number of yellow pixels is as important as having the correct number of blue or green ones. In the next section we will present some of the experimental results achieved.

3 Experimental Results

To test our approach we conducted a series of experiments. As training images, we used some of the early works of Wassily Kandinsky. The images where reduced to the size of 96*96 pixels in order to allow a faster evolution. Although our system may use several training images at the same time, we haven't taken advantage of this possibility so far. The results presented in this section concern the evolution of programs that take the lightness channel of an image as input and give the hue channel as output. As fitness we used $H_{a'}+H_{b'}$.

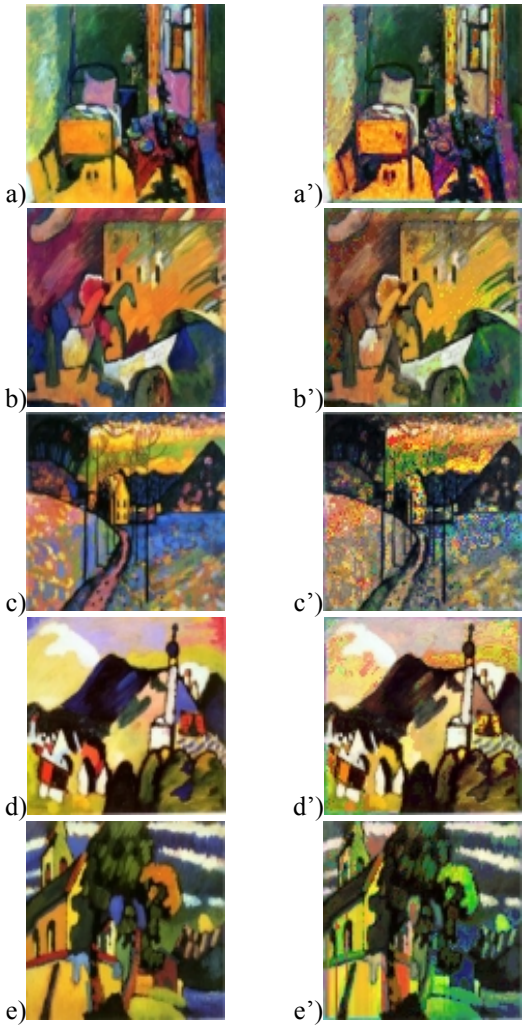

Figure 2: On the left column the original images, on right the images resulting from the application of the best individual of each run to the lightness channel of the training image.

In Fig. 2 we present some of the training images, and the images generated by the evolved programs[1].

It is clear from the results presented that we can achieve images very close to the original ones. However, what is really important to our goal is accessing how well do the evolved programs perform when applied to images not involved in their training, i.e. their generalisation capabilities. In Fig. 3 we present some results achieved with this mode of operation.

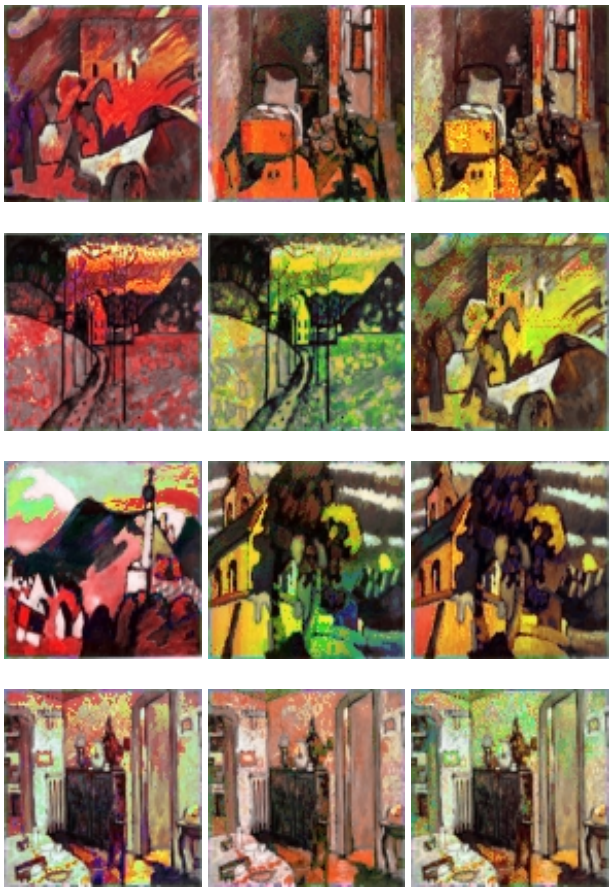

Figure 3: The images on the first column result from the application of the program that generated the image a' from Fig. 2; on the second column, images resulting from the application of b'; on the third column images resulting from the application of c'.

We consider these results to be extremely promising. Some of the colourings presented in Fig. 3 are quite close to the original ones and, additionally, in some cases, although significantly different, they are still interesting colourings. In order to allow a more equitable assessment of the results, we don't present the original image corresponding to the fourth row of Fig. 3.

4 Conclusions and further work

In this paper we presented our ongoing research whose goal is to learn to colour greyscale images from a set of training instances. This effort is part of a wider research project that aims at building a fully autonomous constructed artist.

The results achieved so far concern, mostly, the evolution of programs that generate the hue channel from the lightness one. The experiments performed on the evolution of programs to generate the saturation channel, seem to indicate that this task is quite simpler. Nevertheless, the replacement of the original saturation channel by one generated through a program will unavoidably imply the introduction of further noise. At the time of writing we can't access exactly how much this will affect the quality of the generated colourings (although we foresee little impact).

There is also the need to conduct additional experiments, specially to use several training images simultaneously, which will hopefully increase the generalization capabilities of the evolved programs. Another aspect that can be improved is the fitness function, we are currently altering it so that it also takes into account the vicinity relations between areas of colour, once again the idea is to promote the evolution of programs that assign colour based in more general concepts.

A final word goes to other types of application of the proposed techniques, for instance, the colouring of black and white movies. Assuming that one of the frames is coloured (either by hand or by some other method) it should be possible to evolve a program that gives correct colours to the surrounding frames.

Acknowledgements

This research project was approved by FCT (Fundação para a Ciência e Tecnologia) and POSI (Programa Operacional "Sociedade da Informação"), and is partially funded by FEDER, project n. POSI/34756/SRI/2000.

We would also like to thank the blind reviewers of this paper, which provided precious remarks not only for the paper in question but also for future research.

[1] A colour version of the paper can be found in: http://www.dei.uc.pt/~machado/research/research.htm

References

Banzhaf, W., Nordin, P. and Francone, F. D. Why introns in genetic programming grow exponentially, *Workshop on Exploring Non-coding Segments and Genetics-based Encodings*, ICGA, East Lansing, MI, USA, 1997.

Banzhaf, W., Nordin, P., Keller, E. and Francone, F. D. *Genetic Programming – An Introduction*, Morgan Kaufman, 1998.

Cohen, H., The Further Exploits of AARON, Painter, SHR, *Constructions of the Mind*, Vol. 4, Issue 2, 1995.

Cohen, H., Colouring Without Seeing: a Problem in Machine Creativity, *AISB Quarterly*, 102:26-35, 1999.

Fukunaga, A. Stechert, A. Mutz, D. A Genome Compiler for High Performance Genetic Programming, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 86-94, Morgan Kaufmann, 1998.

Kandinsky, W., *On the Spiritual in Art*, republished by Dover publications in 1997, 1911.

Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

Machado, P., Dias, A., Cardoso, A., GenCo – A project Report. *Proceedings of the Third International Symposium on Artificial Intelligence and Adaptive Systems (ISAS'2001)*, La Havana, Cuba, 2001.

Machado, P., Cardoso, A., All the truth about NEvAr. *Applied Intelligence, Special issue on Creative Systems*, Bentley, P. Corne, D. (eds), Vol. 16, Nr. 2, pp. 101-119, Kluwer Academic Publishers, 2002.

Nordin, P. A compiling genetic programming system that directly manipulates the machine-code. *Advances in Genetic Programming*, Kenneth E (Ed.), pp 311-331, MIT Press, 1994.

Nordin, P., Banzhaf, W. Complexity Compression and Evolution, *Genetic Algorithms: Proceedings of the Sixth International Conference*, ICGA95, pp. 310-317, Morgan Kaufmann, 1995.

Sims, K., Artificial Evolution for Computer Graphics, *ACM Computer Graphics*, Vol. 25, pp. 319-328, Addison-Wesley: Boston, MA, 1991.