Automatic Invention of Fitness Functions with Application to Scene Generation

Simon Colton

Department of Computing, Imperial College, London
180 Queens Gate, London, SW7 2RH, UK
sgc@doc.ic.ac.uk

Abstract. We investigate the automatic construction of visual scenes via a hybrid evolutionary/hill-climbing approach using a correlation-based fitness function. This forms part of The Painting Fool system, an automated artist which is able to render the scenes using simulated art materials. We further describe a novel method for inventing fitness functions using the HR descriptive machine learning system, and we combine this with The Painting Fool to generate and artistically render novel scenes. We demonstrate the potential of this approach with applications to cityscape and flower arrangement scene generation.

1 Introduction

The Painting Fool (`www.thepaintingfool.com`) is an automated artist which has been designed primarily to produce interesting and aesthetically pleasing artworks, but also to test certain high level theories about how people perceive software as creative or not, within the computational creativity paradigm of Artificial Intelligence. The Painting Fool has so far been employed to take digital images and ground truths about those images, segment the images into paint regions and render them stroke-by-stroke by simulating natural media such as pencils, pastels and acrylic paints in a diverse range of styles.

We describe here an addition to The Painting Fool's capabilities. In particular, we describe a novel application where the input to The Painting Fool is not images, but rather a high-level description of a complex scene such as a cityscape. It then uses a hybrid hill-climbing/evolutionary approach to generate a segmentation according to the scene specification, and proceeds to render this in an artistic style. As described in section 2, generic scenes are specified as a set of elements and a set of desired correlations between aspects of the elements. The hybrid evolutionary/hill-climbing search methods we tested for constructing such scenes are described in section 3. Inspired by [2], we took this application to the meta-level by using the HR machine learning system [4] to invent novel fitness functions for scene elements, as described in section 4, with some illustrative results given in section 5. To conclude, we argue that with this project, and others, The Painting Fool has exhibited behaviour which is skillful, appreciative and imaginative, and as such, could be described as creative.

2 Problem Description

We are interested in the problem of automatically generating scenes which potentially contain hundreds of similar scene elements. We use a simple representation of scenes as an ordered list of *scene elements*. To describe a generic scene to the system, the user/developer must provide the following:

– high level details of the scene, namely the pixel height and width of the scene and how many scene elements it should contain.
– a specification of a *scene element* in terms of a set of numerical attributes and a range of values for each attribute.
– an (optional) set of attributes to be calculated from the specifying attributes of a scene element, with suitable calculation methods.
– a set of desired correlations which the numerical attributes must conform to, and a weighting indicating the importance of each correlation.
– a method for using the attributes of each element to transform it into a set of colour segments as used by The Painting Fool.

With respect to the desired correlations that the user specifies, we calculate the Pearson product-moment correlation of two numerical variables v_1 and v_2 to approximate how they are related. This produces a value of: 1 if the variables are positively correlated, i.e., if v_1 increases, then so does v_2; -1 if the variables are negatively correlated, i.e., if one increases then the other decreases and vice-versa; and 0 if they are not correlated. In addition to specifying which two attributes of scene elements should be assessed by correlation, the user also supplies the correlation value they require. For instance, if the user requires two attributes to be completely positively correlated, they specify that the correlation should score 1, but if they want the two attributes to have some, but not total positive correlation, they might specify 0.5 as the required correlation score.

Using the terminology of Ritchie [10], our *inspiring example* has been scenes approximating Manhattan-style skylines. To achieve such scenes, we described them as being 1000×200 pixels in dimension with 300 elongated rectangular scene elements. Each element is described by the following attributes: (i) x-coordinate, of range 0-1000 (ii) y-coordinate, of range 60-100 (iii) width, of range 5-20 (iv) height, of range 20-100 (v) hue, of range 0-360 (vi) saturation, of range 0-1 and (vii) brightness, of range 0-1. We also provided methods to calculate (a) the *edge distance*, i.e., the minimum distance of the scene element from the left or right hand side of the scene and (b) the *depth* of the scene element, which is the number of scene elements that are both earlier than the element in the ordered list and overlap it. In addition, we specified 7 correlations with equal weights that the scene element attributes should adhere to:

– the edge distance of a scene element should have a correlation of 1 with the element's (i) width, (ii) height, (iii) y-coordinate and (iv) saturation.
– the depth of a scene element should have a correlation of: (v) 1 with the element's saturation (vi) -1 with the element's height and (vii) -0.5 with the element's brightness.

These correlations were chosen – after some experimentation – so that buildings in the middle of the scene appear bigger than those at the edges, and that elements at the back of the scene are taller – so that they can be seen – and less saturated, which is also true of the scene elements at the edge of the scene. This gives the impression that the buildings at the edges and at the back are further away than those in the centre of the scene (which is similar to the Manhattan skyline as viewed from the Staten Island ferry).

3 Search Methods for Scene Generation

Recall that the user supplies a set of n correlation specifications, which can be thought of as quadruples $\langle u, v, r, w\rangle$, where r is the required value that the Pearson product-moment calculation $p(u, v)$ should return for ordered lists of real-numbered values u and v, and w is a weight such that $\sum_1^n w_i = 1$, with the weights indicating the relative importance of the correlation. These quadruples can be expressed as a fitness function for scenes S which returns values in the range 0 to 1 by calculating:

$$f(S) = \sum_i^n w_i \left(1 - \frac{|r_i - p(u_i, v_i)|}{max\{|1 - r_i|, |-1 - r_i|\}}\right)$$

This scores 1 if all specified pairs of scene element attributes in a scene are perfectly correlated as per the specification, and 0 if their correlation score is as far away as it can be from the specified value.

We first looked at an evolutionary approach to generating scenes which maximise the fitness function. As in a standard evolutionary approach, the user specifies the population size, number of generations, mutation rate and crossover method (1-point or 2-point). An initial population of scenes each containing the requisite number of scene elements is then generated randomly. This is done by choosing values for each scene element attribute in each scene randomly from those allowed by the user-specified range. The fitness $f(S)$ for each scene S in the population is calculated, and S is given an overall score $e(S)$ by dividing $f(S)$ by the average of f over the entire population. $\lfloor e(S)\rfloor$ copies of S are placed into an intermediate population, with an extra one added with a probability of $e(S) - \lfloor e(S)\rfloor$. Pairs are chosen randomly from the intermediate population and an offspring is produced from them via crossover. Doing so is a relatively straightforward matter, as each scene is an ordered list of scene elements of a fixed size, hence sequential blocks of scene elements are simply swapped from the parents into the offspring. Offspring are mutated and added to the new population until there are the requisite number of individuals in the population. Each scene element in an offspring is chosen for mutation with a probability proportionate to the mutation rate. To perform a mutation, the chosen scene element is removed, then a completely new one is generated and inserted into the ordered list at a random position.

We also experimented with a hill-climbing approach. Here, a random scene is generated as above and each scene element is altered in turn in a single pass.

This is done by choosing a random value for each numerical attribute (and position in the ordered list) and checking whether this improves the fitness of the entire scene. If so, the new value is kept, but if not, the original value is re-instated. The user specifies a repetition factor for hill-climbing. This dictates the number of times a random value is chosen for each attribute of each scene element, and is usually in the range 1 to 100. With initial experimentation using simple cityscape scene descriptions, we found that both approaches were able to produce scenes with fitness over 0.9 in search sessions lasting only a few minutes. However, with our inspiring example of Manhattan cityscapes, which uses 7 correlations, we found that we could only achieve fitnesses between 0.8 and 0.9, using population sizes and generation numbers that resulted in run-times over 10 minutes. Moreover, on inspection of the scenes produced with fitness less than 0.9, we found that they were not of sufficiently high quality for our purposes. Hence, we also experimented with a hybrid approach, whereby an evolutionary approach is first attempted, and the most fit individual ever recorded becomes the basis of a hill-climbing search. We found that this performed better than both stand-alone methods.

To determine the optimal search strategy in terms of the highest achievable fitness in a reasonable time, we searched for solutions to the Manhattan example with 56 different search setups. For the evolution-only approach, we varied population size (100, 500 and 1000), number of generations (100, 500), mutation rate (0, 0.1, 0.01 and 0.001) and crossover method (1 and 2 point). For hill-climbing, we tested 1, 5, 10 and 100 repetition steps. We then examined the results and chose four hybrid setups to test. The best 20 setups are presented in figure 1 along with the fitness achieved and the time taken (all experiments were performed on a 2.4Ghz machine running Mac OS X).

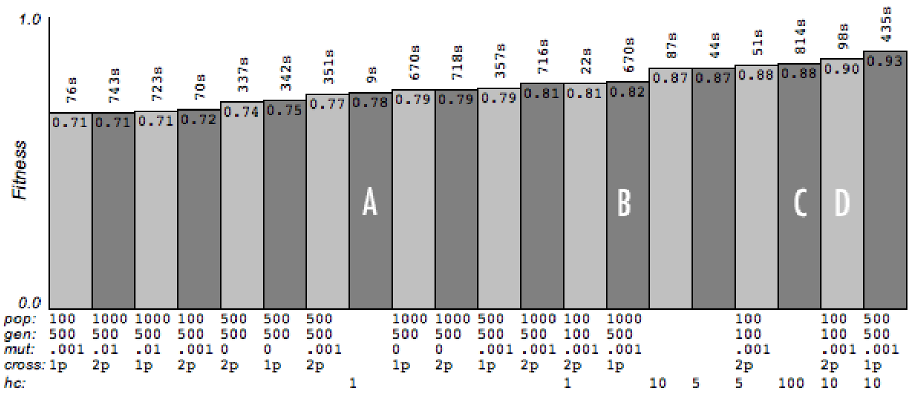

Fig. 1. Best 20 search setups for generating Manhattan-style scenes. Time taken in seconds and fitness are indicated at the top of each bar, with the search setup parameters given below each bar: pop=population size, gen=number of generations, mut=mutation rate, cross=crossover method [(1p)oint and (2p)oint], hc=hill-climbing repetition.

There were some interesting findings in these results. Firstly, we noted that anything but a 0.001 mutation rate resulted in premature convergence, which – as suggested by a reviewer – is probably due to the roughness of the fitness landscape. We found little discernable difference between 1 point and 2 point crossover, and as expected, with evolution only, the fitness achieved was largely proportional to the population size and number of generations. The quickest setup (labelled setup A in figure 1) achieved a fitness of 0.78 in only 9 seconds, using hill-climbing with repetition factor 1. We found that hill-climbing with a repetition rate of 100 achieved a fitness of 0.88, but took more than 10 minutes to achieve (setup C). In comparison, the best evolved (non-hill-climbing) scene had a fitness of 0.82 and took 670 seconds to produce (setup B). In contrast, an evolutionary approach with population size 100 for 100 generations followed by a hill-climbing session with repetition factor 10 achieved fitness 0.9 in only 98 seconds (setup D). It seems likely that we could achieve better results with different settings, and perhaps by using an iterative hill-climbing approach. However, as our main focus is on automatically generating fitness functions, and setup D performs adequately for that task, we have not experimented further yet.

In figure 2, we show the scene generated using search setup D, which achieved a fitness of 0.9. We see that, while there are a few outliers amongst the scene elements, the desired properties of the scene are there. That is, the buildings at the left and right of the scene are smaller in both width and height, less saturated and higher, which gives them the appearance of distance. Also, the buildings at the back of the scene are taller, less saturated and slightly brighter, again giving the impression of distance. In figure 2, we also present two rendered versions of the cityscape. The first is rendered using simulated coloured pencil outlining (with a reduced palette of urban colours) of the buildings over a simulated pastel base on art paper. The second is rendered using simulated acrylic paints over a pastel base, on primed canvas, giving a slightly three dimensional effect.

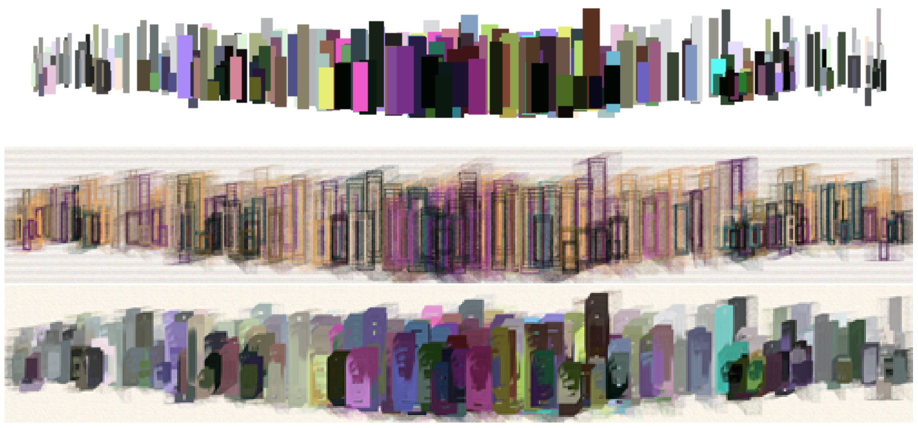
Fig. 2. Evolved setup D cityscape scene, rendered with: block shapes; simulated pastels and pencils; and simulated acrylic paints.

4 Automatic Invention of Fitness Functions

One of the defining aspects of Cohen's AARON program [9] is that it invents scenes. This has been a motivation for development of the scene generation abilities of The Painting Fool. However, with the approach described above, it is difficult to state that The Painting Fool fully invents the scenes it paints, because the user must specify both the scene elements and some required properties of the scene as a whole, i.e., by specifying a fitness function in the form of a set of weighted correlations. We have also been motivated by Buchanan [2] and others in their opinion that meta-level reasoning is essential for creative behaviour. For this reason, we decided to use the HR system to invent fitness functions so that the user is able to specify only the types of elements a scene should contain.

The HR system is a descriptive machine learning program which starts with minimal information about a domain and generates examples, invents concepts which categorise the examples, makes hypotheses which relate the concepts, and proves these hypotheses using third party reasoning systems. HR has been described extensively elsewhere, e.g., [4, 7]. For our purposes here, we need to know only that it is able to take background concepts which describe objects of interest and invent new concepts via a set of generic production rules. For instance, in number theory [3, 6], HR is given just the ability to multiply two numbers together. It then invents the concept of divisors, using its *exist* production rule, then the concept of the number of divisors of an integer, using its *size* production rule, then the concept of integers with exactly two divisors (prime numbers), using its *split* production rule. In this way, HR invented novel integer sequences which, in the terminology of Boden [1], were H-creative. HR has been used in a number of other mathematical domains, in particular finite algebras [11].

To enable HR to invent fitness functions for scene generation, we made the objects of interest in theory formation sessions the scenes, with each object described by its scene elements, and each scene element described by a set of numerical attributes. Hence, the background concepts supplied to HR were essentially the set of attributes of scene elements specified by the user. In addition, we have extended HR to work with floating point data. To do so, we wrote a floating-point version HR's existing *arithmetic* production rule. The new version is able to take, for instance, the concepts describing the height of scene elements and the x-coordinate of scene elements and invent the concept of the height *plus* the x-coordinate of scene elements. In a similar manner, HR can subtract, multiply and divide pairs of floating point concepts. We also implemented two new production rules. Firstly, the *correlation* rule is able to take two pairs of floating point concepts and produce the concept of the correlation between the two. Secondly, the *float-summary* production rule is able to take in a single floating point concept and produce a concept with summary details about the values, namely the minimum/maximum/average value, the smallest difference between two values and the range of the values. Note that, while we have described these new rules in terms of scenes and scene elements, they are generic and would work in any other domain with floating point background concepts.

```
Input
S: scene overview specifications (number of scene elements, scene width & height)
E: scene element attributes and ranges for the attributes
R: rendering specifications
Algorithm
1.  Using $E$, $R$ and $S$, TPF generates five scenes randomly with 10 elements
2.  TPF translates the scenes into a HR input file
3.  TPF invokes HR to produce a theory using 1000 steps
4.  HR translates its theory into a Java class, $J$, and compiles it
5.  TPF constructs a random population, $P$ of 100 scenes according to $S$
6.  while ($maxfitness(P,F) < 0.4$ or $maxfitness(P,F) > 0.8$)
        TPF builds a fitness function $F$ by choosing 3 to 6
        correlation concepts from $J$ and giving them equal weights
7.  TPF evolves a best scene, $B$, according to $F$ using setup D (see section 3)
8.  TPF hill-climbs to improve $B$ using repetition factor 10 (setup D again)
9.  TPF translates $B$ into a segmentation, $G$
10. TPF renders $G$ according to $R$
```

Fig. 3. Overview of the interaction between HR and The Painting Fool (TPF).

In figure 3, we describe in overview how a combination of The Painting Fool and HR were used to automatically construct a fitness function, use this to search for a scene which maximises the fitness, and then render the scene. Note that, to keep HR's run-time to around two minutes, it is only supplied with 5 scenes of 10 elements, which is enough for it to form a theory. Note also that, in step 6, The Painting Fool builds a fitness function using a Java class that HR has generated. The Java class is able to take a scene and calculate a fitness for it. The Painting Fool constructs the fitness function as an equally-weighted sum of between 3 and 6 correlation concepts randomly chosen from HR's theory. It checks that the best fitness in a randomly generated population of 100 scenes is above 0.4 (so that there is a chance of evolving a scene to above 0.9 fitness), and below 0.8 (which ensures that a random scene will not be output). If this is not true of the fitness function it has constructed, it tries again until it succeeds. As an example, the first cityscape scene generated in a session described in the next section was generated using a fitness function with the five equally weighted correlations below. The scene generated using this is given in figure 4.

cor(brightness, x-coord) = 1	
cor(y-coord - x-coord, brightness) = -1	cor(brightness, height) = 1
cor(y-coord * x-coord, height) = 1	cor(saturation, height) = -1

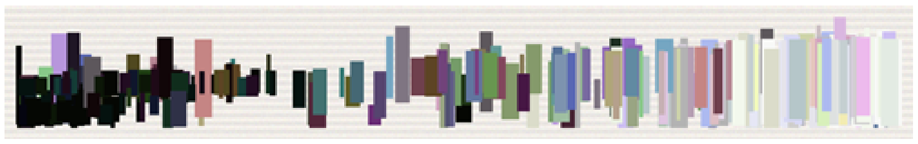

Fig. 4. Scene generated with an automatically invented fitness function.

5 Illustrative Results

We present here two scene generation sessions with The Painting Fool/HR. Firstly, we used the cityscape scene specification as above, and the algorithm in figure 3 to generate a fitness function. We repeated this 10 times, with the results from the session given in figure 5, alongside two randomly generated scenes for comparison. In each of the 10 generated scenes, there are at least two noticeable patterns, e.g., in scene E, the buildings are more colourful at the edges and wider on the left than on the right. In scenes B and G, two distinct clusters of buildings were generated, which we did not expect, and in scene C, the fitness function forced some buildings to be placed vertically on top of each other, which was also not expected. The fitness of the scenes were in the range 0.88 to 0.94.

In the second session, we tested the flexibility and ease by which the system can be given a new task, using arrangements of flowers, as this is quite different to cityscapes. We chose 17 closely cropped digital images of flowers, and used The Painting Fool to generate the segmentations of these. We then described the generic scene in terms of scene elements with the following properties: size (30 to 150), x and y coordinates (both 0 to 400), and flower number (1 to 17). We wrote code so that when The Painting Fool translates the scenes into segmentations, it simply retrieves the segmentation for the appropriate flower number from file. We also wrote code able to calculate (i) the distance of the scene element from the centre of the scene and (ii) the position in the ordered list that the scene element appears at. To test the scene specification, we used a fitness function with three correlations: cor(centre-distance, flower-number) = 1; cor(centre-distance, size) = -1; cor(centre-distance, pos) = -1. This meant that the scenes had flowers around the edge which were smaller (with the smaller ones being painted later than the larger ones), and that the flowers portrayed changed from the centre to the outside of the scene.

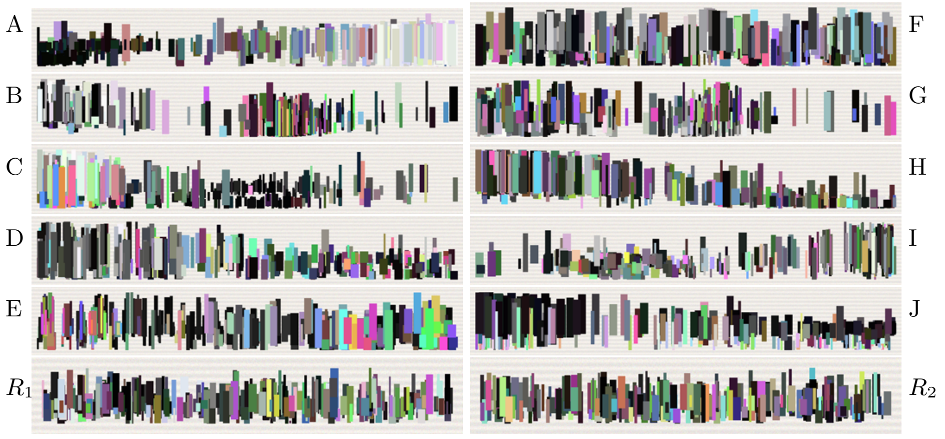

Fig. 5. Cityscape scenes generated in session 1 using invented fitness functions (A to J), compared to random cityscapes (R_1 and R_2).

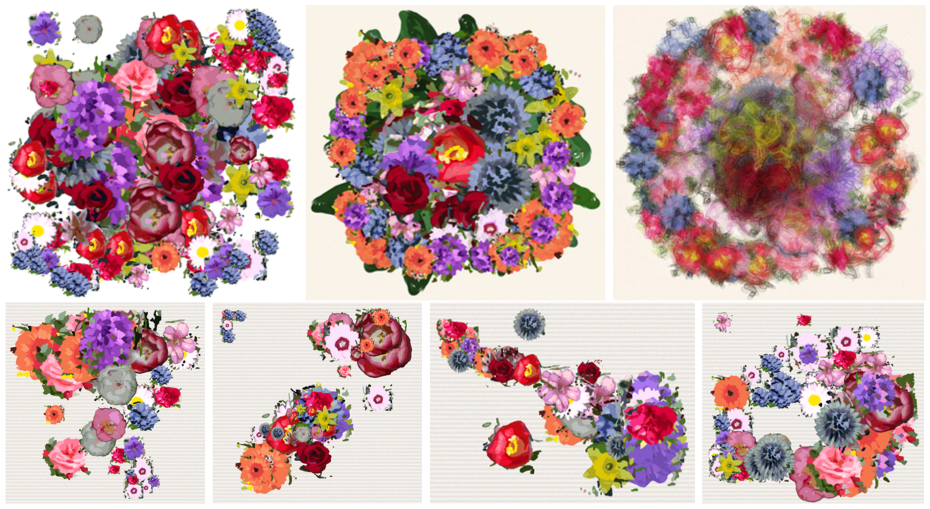

Fig. 6. Cartesian flower arrangement scene; polar coordinate scene 1, rendered with simulated acrylic paints onto a painting of leaves; polar coordinate scene 2, rendered with simulated pastels and pencils; four invented flower arrangement scenes.

We ran a search using setup D to generate a scene with 150 flowers in it, with the resulting scene given in figure 6. We also repeated this experiment twice using polar instead of Cartesian coordinates for the scene elements, to give a circular arrangement of the flowers. In figure 6, we present artistic renderings of the two polar coordinate scenes. Using the Cartesian setup, we then ran the automatic invention of fitness function routine 10 times, but to produce scenes with only 50 elements, and we chose the four most interesting to show in figure 6 (an entirely subjective choice by the author). As with the invented cityscapes of figure 5, each of the four scenes clearly exhibits a pattern. However, of the six other scenes from the session (which are not shown), with four of them, we could discern no obvious scene structure. These scenes scored less than 0.8 for fitness, and on inspection, this was because the fitness functions needed to achieve contradictory correlations. With fewer attributes to seek correlations between in this application than in the previous one, the likelihood of generating such contradictory fitness functions was higher. We aim to get The Painting Fool to avoid such cases in future. Including the time taken to find and segment the flower photographs and for us to write the necessary code for attribute calculation and segmentation translation, the entire session took around 4 hours.

6 Conclusions and Further Work

We have described a hybrid evolutionary/hill-climbing approach to the construction of scenes, via a correlation based fitness function, and a method which uses the HR system to invent such fitness functions. To the best of our knowledge, while this work fits into the context of co-evolution of fitness functions such as in [8], it is the first time a descriptive machine learning system has been used to invent a fitness function. We demonstrated this for two types of scene, namely cityscapes and flower arrangements. As suggested by two reviewers, we have compared the scenes generated using HR's invented fitness functions against ones generated using an equal weighting of randomly chosen correlations. While these scenes showed clear patterns, as the randomly generated fitness functions are a subset of those produced by HR, the scenes lacked variety somewhat. We plan to undertake more extensive experimentation to qualify these findings. We have used only a fraction of HR's abilities in the experiments here, and we plan to use HR to invent more sophisticated fitness functions, and to investigate using HR in evolutionary problem solving in general. We plan many other improvements, including: non-correlation based fitness functions; multiple subscenes; post-processing of scenes to remove outliers; 3d scene generation; and – as suggested by the reviewers – the generation of fitness functions from exemplar scenes, and the usage of a multi-objective evolutionary approach.

As described in [5], we believe that software exhibiting skill, appreciation and imagination should be considered creative, and we are building The Painting Fool along these lines. In two other projects (the Amelie's Progress gallery and the Emotionally Aware application described on `www.thepaintingfool.com`), The Painting Fool has exhibited behaviour which could be considered appreciative. In enabling it to invent fitness functions and generate scenes using them, we have implemented some more imaginative behaviour, and we hope to have added a little to the case that the software is creative in its own right.

Acknowledgements

We wish to thank the anonymous reviewers for their excellent comments, which improved this paper and suggested many interesting avenues for further work.

References

1. M Boden. *The Creative Mind.* Weidenfeld and Nicolson, 1990.
2. B Buchanan. Creativity at the meta-level. *AI Magazine*, 22(3):13–28, 2001.
3. S Colton. Refactorable numbers - a machine invention. *J. Int. Sequences*, 2, 1999.
4. S Colton. *Automated Theory Formation in Pure Mathematics.* Springer, 2002.
5. S Colton. *Creativity versus the Perception of Creativity in Computational Systems.* In *Proc. of the AAAI Spring Symposium on Creative Intelligent Systems*, 2008.
6. S Colton, A Bundy, and T Walsh. Automatic invention of integer sequences. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, 2000.
7. S Colton and S Muggleton. Mathematical applications of Inductive Logic Programming. *Machine Learning*, 64:25–64, 2006.
8. M L Maher and J Poon. Co-evolution of the fitness function and design solution for design exploration. In *IEEE Int. Conf. on Evolutionary Computation*, 1995.
9. P McCorduck. *AARON's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen.* W.H. Freeman and Company, 1991.
10. G Ritchie. Assessing creativity. In *Wiggins, G., ed., Proceedings of the AISB'01 Symposium on AI and Creativity in Arts and Science*, pages 3–11, 2001.
11. V Sorge, A Meier, R McCasland, and S Colton. Automatic construction and verification of isotopy invariants. *J. of Automated Reasoning*, Forthcoming, 2007.