Evolutionary Computation in Painting Systems

Pertti Perusjamppa

Seminar report

UNIVERSITY OF HELSINKI
Department of Computer Science

Helsinki, January 28, 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Pertti Perusjamppa

Työn nimi — Arbetets titel — Title
Evolutionary Computation in Painting Systems

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Seminar report	January 28, 2015	15

Tiivistelmä — Referat — Abstract

Although a huge volume of image altering filters and image creation software exists, most of these end up needing a human interaction to produce appreciated results. In this report, image creation systems are studied from the aspect of computational creativity. Computational creativity studies creative systems and their behaviour, most notably, how systems can produce artefacts which are suprising and appreciated. Two image creation systems utilising evolutionary computation to reach appreciated artefacts are presented. Both of the systems are evaluated based on Wiggins' model; a mathematical model created to help in evaluation of creative systems. The systems are observed to exhibit creative traits to some degree, but the limitations in the system domains play a key role in restricting the emergent behaviour.

ACM Computing Classification System (CCS):

- **Computing methodologies~Optimization algorithms**
- **Computing methodologies~Discrete-event simulation**
- Computing methodologies~Non-photorealistic rendering
- Applied computing~Fine arts

Avainsanat — Nyckelord — Keywords
genetic algorithm, genetic programming, creative systems, non-photorealistic rendering, visual arts

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Using algorithms in image rendition is widely studied topic, and it is not surprising that a plethora of visually pleasing filters and other algorithmic image manipulation techniques exist. However, most of these techniques generate consistent and similar alterations to the images (given the parameters) in successive executions. Although this kind of behaviour is usually desired by the normal users, it does not end up being surprising or creative. In fact, from the viewpoint of creativity, the filters and algorithms end up being mostly uninteresting, and the creativeness of the image manipulation process stems from the user's interaction.

The uninterestingness of conventional methods raises a question: when does a system, e.g., an image creation or alteration program, truly express creative behaviour? The researchers from the computational creativity field have tried to answer this question by formalising the model in which the system operates, and defining the characteristics of the systems which are genuinely creative. One such attempt is made by Wiggins [13] who describes his motivation: *"The aim is to move towards a model which allows detailed comparison, and hence better understanding, of systems which exhibit behaviour which would be called "creative" in humans"*. By using Wiggins' model it is possible to, if not exactly evaluate by assigning a single value, at least have an estimation of systems' general creative characteristics.

In order for the system to be creative, it has to be able to create artefacts which are both novel and have value. Margaret Boden has described three different ways for the systems to reach these artifacts: making unfamiliar combinations of familiar objects, exploring the conceptual space, and transforming the conceptual space [1]. Boden finds the transformational creativity to have the most value, but Wiggins argues that by formalising the model of creative systems the transformational creativity can be seen as exploration at the meta-level; hence, simplifying the model [13].

In this report, two systems which utilise evolutionary computation to create images are presented, and evaluated against Wiggins' model. First system, Photogrowth, is an image rendering tool, which uses ant colonies and evolutionary engine to acquire non-photorealistic renderings of given input images [7, 4]. The second system, created by Machado et al. [6, 3], utilises image classification system and genetic programming to create figurative images in four different categories based on mathematical expressions.

The rest of the report is organised as follows. First, a short philosophically inspired journey is taken to consider how human painters operate. In Section 3 the fundamentals of evolutionary computation, and, especially, genetic algorithms are

described. Also some notions about the relation of genetic algorithms and Wiggins' model are presented. Then, the two image creation systems are discussed in more detail, and the report continues to evaluate their traits of creativeness in Section 6. The report ends in conclusions with a few modest notions about the possibilies of how the creativity of the systems could be enhanced.

2 What Does a Painter Do?

The question in this section's title is of some note when trying to form an overall picture of how image creation systems could exhibit creative behaviour. Although one could argue that the imitation of humans can not contain any intrinsic value when evaluating arbitrary system's creativity, it implicitly leads to another question. In which scale, domain or "reality" does the system operate? For example, Mr. Pixel could say that a painter is an agent who creates valued artefacts by coloring pixels in 2D-space with a strategy which is neither fully random nor completely deterministic. On the other hand, Sally Social could define a painter as an agent who lives in a society consisting of a population of agents. The agents in Sally's world would not only create visually pleasing artefacts, but also have other behaviours, especially, social interaction with other agents.

When comparing Mr. Pixel's and Sally Social's realities, or universes the agents are living in, it is clearly seen that Mr. Pixel's reality could be only one behaviour of an agent in Social's reality. An agent in Pixel's reality is more forcefully constrained in its conceptual space, than the agents belonging to a social universe where they can interact with their surroundings more freely. In this sense, if a painter is described as an agent who only continuously creates 2D-images, and the whole universe revolves around it, it would be quite unexpected for the agent to discover completely new conceptual aspects, e.g. adding a third dimension to the artefacts it creates.

Disregarding the simplified answers of Pixel and Social, the question most likely has been asked, and answered, repeatedly in history. Simon Colton, the leader of the team contributing to the perhaps most famous painting software, The Painting Fool, describes the answering process as a way for the team to focus on right areas while developing the software [2]. Colton proceeds to formulate the answer with seven remarks:

1. Makes marks on a canvas.
2. Represents objects and scenes pictorially.
3. Paints scenes in different styles.

4. Chooses styles in a meaningful way.
5. Paints new scenes from imagination.
6. Invents scenes for a purpose.
7. Learns and progresses as an artist.

Straightforward examination of the answer seems to indicate, that even Colton leaves the social aspect out, however, it can be considered to be implicitly inherent in the last note as the actual learning and progression processes are not defined. Also, dissection of the answer in relation to Wiggins' model[1] results in interesting notions. The purpose mentioned in the 6th remark seems to point to $\mathcal{E}$, but it holds a deeper meaning for an agent who is more than an assembly line for paintings, since the motivation to paint, and what to paint, can arise from several sources. The styles mentioned in points 3 and 4 are something that not only affect the appreciation, but also how the search space is explored, and what kind of restrictions are applied. In this context, a painter exploring new styles could be seen as $\mathcal{R}$- or $\mathcal{T}$-transformationally creative, however, this should be viewed with caution as the definitions rely on the conceptual space construct.

3 Evolutionary Computation

The idea of using evolution to solve computational problems was developed in the 1950s and the 1960s in multiple independent works [11]. For example, John Holland constructed *genetic algorithm*, Lawrence Fogel defined *evolutionary programming*, and Ingo Rechenberg described *evolution strategies*. However, by the 1990s the subfield of artificial intelligence had stabilized enough for the methods to be accepted as different aspects addressing the same underlying general principles.

Today, the term evolutionary computation is used to encapsulate a group of strategies that solve optimisation tasks through iterative processes, e.g., by a development in a population or a swarm (thus, roughly the same group of strategies is sometimes called *population-based methods*). The strategies are grouped under the same term because they utilize different mechanisms found in nature, and – most notably – in evolution. These strategies are often used in tasks, where either no

[1] In short, Wiggins' model is a way of seeing creativity as a search. It consists of four different mathematical constructs $\mathcal{U}, \mathcal{R}, \mathcal{E}$ and $\mathcal{T}$. $\mathcal{U}$ is the universe containing all *possible* things, $\mathcal{R}$ is a rule set which restricts the universe to contain only acceptable objects in order to form a conceptual space $\mathcal{C}$. Evaluation rules $\mathcal{E}$ tell how objects are appreciated, and traversal rules $\mathcal{T}$ tell how the search space is explored. See [13] for details.

exact algorithms are known, or they are computationally too complex and finding a (possibly) suboptimal solution is still desirable.

Particularly, this report discusses about two evolutionary computation methods: *genetic algorithms* (GA) and *genetic programming* (GP), the latter of which is a special case of the former. Genetic algorithms exploit a general search heuristic that tries to mimic the natural selection in a population. Each individual in the population has a set of genes (or chromosomes), which are manipulated in order to acquire individuals with high *fitness*. The algorithm consists of four different phases: *initialisation*, *selection*, *genetic operators* and *termination*.

In its most basic form, the genetic algorithm's phases are executed as follows. In initialisation, the population is first constructed. It can be done randomly, or using any strategy deemed appropriate. Next, in the selection phase, the fitness of each individual is evaluated by some criteria and the individuals with low fitness are discarded using a *selection strategy*. The genetic operators are then applied to the remaining population in order to regrow the population to its original size. The resulting individuals are then treated as the next generation of the population. Also, sometimes an elitist strategy is deployed – the most fittest individuals are moved as is to the next generation. Now, the algorithm moves back to the selection phase where it ranks each individual again. The selection and genetic operators are then successively applied as many times as is needed until the desired termination criteria is met, creating the characteristic iterative process of evolutionary computation.

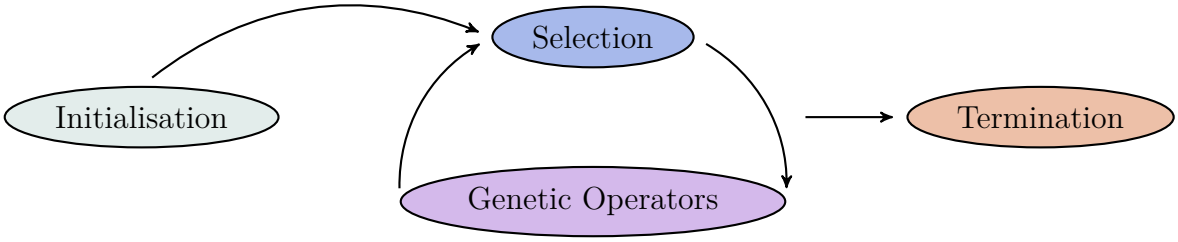


Figure 1: Genetic algorithm's execution phases. First, population is initialised, and then selection and genetic operation strategies are cycled until termination criteria is met.

In a generic GA implementation the individuals, or the genes to which the genetic operators are applied, are not thematically restricted. Instead, they can contain any information which is needed in the optimisation task. On the other hand, this is not the case in genetic programming where each individual is a computer program, e.g. a mathematical expression. The genetic operators applied in the mutation phase usually contain at least *point mutation* and *crossover* (recombination). Point

mutation changes a single gene in a single individual, and crossover switches a gene, or a set of genes, between two individuals. Both of these are done by a certain probability: *mutation probability* and *crossover probability*, respectively. Usually crossover probability is set quite high, and applied to individuals with high fitness to direct the search to promising directions. The mutation probability is usually low and applied more randomly to introduce new genetic material to the population. The exact implementation of the operators and their probabilities vary between domains, and is a crucial design factor when tuning the quasi-random search of the GA.

In comparison to Wiggins' model [13], creative systems applying genetic algorithms as main search strategies have few implicit analogies. Importantly, a genetic algorithm *can* be indifferent of the conceptual space, as it does not have to have a stance on actual construction of artefacts; the same individual can be interpreted differently in order to construct artefacts belonging to varying concepts. However, this is not the case in real implementations, as the fitness function's main role is the evaluation of value, represented by evaluation rules $\mathcal{E}$ in Wiggins' model, thus it is instrumentalised to interpret traits collected from created artefacts. Furthermore, the traversal rules, $\mathcal{T}$, are the implemented genetic operators, and to some extent the evaluation criteria, as it tries to guide the search to more promising areas. The allowed value ranges of the mutable genes can be seen to belong either into $\mathcal{T}$ or restriction rules $\mathcal{R}$, depending on the definition of the conceptual space C.

4 Photogrowth - Ant Colony Paintings

This section describes the interactive painting tool, Photogrowth, first introduced by Machado and Pereira [7]. The system revolves around two main components: painting algorithm and evolutionary engine. The painting algorithm uses ant colonies to create non-photorealistic renderings of the input image, and evolutionary engine allows evolution of different ant colonies in the search for high fitness renderings. The system allows the user to initialise the evolutionary engine with several parameters considering the ant colonies. In interactive sessions described in [7], the user is in charge of the evolutionary process by evaluating each rendered image. After pleasing ant colonies are found, they can be saved and used on other images.

Machado and Amaro [4] use the same framework to experiment on fitness functions in order to automatise the evolutionary process. After coarsely outlining the overall image generation process as described in [7], some effort is put into understanding couple of experimented fitness functions, and the differences they create in population evolution.

4.1 Ant Colonies

The ant colony for the painting algorithm consists of a population of ants, each with following individual features: color, position, energy and deposit transparency. Other ant traits, i.e. genotype, are shared with the population, and are subject to mutation. The ant population development is simulated with an iterative algorithm, in which the ants live in a 2D-world consisting of living and painting canvas, one on the other – each ant is always in the exact same position on both canvases. The living canvas is the input image, where the ants move around searching for food. The painting canvas is first black, and is used exclusively as a painting medium.

At first, ants are placed on the canvases, and they assume the color present in the same location on living canvas; importantly, the color does not change during the ant's lifetime. From there on, the ants can observe their surroundings with several antennae, all of which point to diffrent directions (with respect to ant's current orientation), and have varying lengths and weights.

Figure 2: From left: an ant with five antennae, living canvas and painting canvas, as seen in [4].

When ant moves to certain position, it "eats" the luminance from the input image trying restore its energy, therefore lowering the luminance left in the image.[2] Afterwards, the ant deposits ink, i.e. draws a circle of its color, to the painting canvas.[3] The movement direction for each iteration and ant is guided by how the ant's antennaes observe luminances. Explicitly, the displacement vector $\Delta\vec{p}$, for an

[2]The actual energy change amount depends on the current luminance in the position, and the *gain* and *decay* values determined by the genotype, i.e. the overall energy of the ant can decrease.

[3]The successive circles are later processed to form continuous trails of paint, see [7] for details.

ant in position (x, y) with a antennae is obtained by

$$\Delta\vec{p} = vel \cdot \sum_{i=1}^{a} \frac{\vec{v_i}}{|\vec{v_i}|} \cdot b((x, y), \vec{v_i}) \cdot w_i, \tag{1}$$

where $\vec{v_i}$, is the vector representing antenna i, and w_i is the weight of the antenna. Furthermore, *vel* is ant's base velocity, and $b((x, y), \vec{v_i})$ is a function returning current luminance on the living canvas in the position $(x, y) + \vec{v_i}$. To mimic inaccuracies in ant's movement and observations, $\Delta\vec{p}$ is further altered by adding noise to the movement angle.

If ant's energy exceeds *birth* parameter encoded in genotype, new ant is spawned in the same location with a portion of the parent's energy, which in return is reduced from the parent ant. The new ant assumes current location's color, and other features are acquired by preturbating parent's feature values within genotype's limits. The ant dies when its energy drops below genotype's *death* parameter.

4.2 Evolutionary Engine

The evolutionary engine handles the evolution of ant colonies. It consists of *a population of colonies*, where each colony has its own parameters for successor's features preturbation limits, birth and death thresholds, antennae weights and vectors, and other useful parameters, such as scaling for the circles drawn into the painting canvas (see [4] for details). The engine uses two point crossover and Gaussian mutation as genetic operators. The selection is done by tournament selection, where groups of colonies are put against each other and the individuals with higher fitness are selected to the next generation with higher probability, paired with an elitist strategy. The termination happens after certain amount of generations (of colonies) has passed.

4.3 Fitness Functions and Results

In relation to the previous work done, the main contribution of Machado and Amaro is the exploration of fitness functions for automating the selection phase of the genetic algorithm. The fitness functions are based on behavioral and image features. The behavioral features are collected during the ant colony simulation and image features are calculated after the simulation has ended. Some features are presented in Table 1.

Machado and Amaro first construct fitness functions based on individual features, but soon move on combining them for more interesting results. More closely, they

Feature	Explanation
Behavioral	
$coverage$	Portion of the image where at least one ant consumed resources.
$avg(std(av))$	Calculate standard deviation for each trail's angular velocity, and then their average.
$avg(avg(av)$	Calculate average of each trail's angular velocity, and take an average of them.
Image	
$inv(rmse)$	The similarity estimation between the input image and the ant painting defined as $inv(rmse) = \frac{1}{1 + rmse(I, O)}$, where $rmse$ is root mean square error, I the ant painting, and O the input image.

Table 1: Some features used in fitness functions for ant colonies.

define 6 fitness functions with different feature sets. In here, due to space restrictions, we will take a look of two closely related ones. The fitness functions f_4 and f_6 are defined as

$$f_4 : avg(std(av)) + inv(rmse) + coverage \tag{2}$$

$$f_6 : avg(std(av)) + inv(rmse) + coverage - avg(avg(av)), \tag{3}$$

where named functions are defined as described in Table 1.

Machado and Amaro explain the motivation behind the function constructions as a way of controlling the line direction. The $avg(std(av))$ is used to appreciate paintings where line direction changes often. The $inv(rmse)$ and $coverage$ can be seen as generally desirable traits which are used to promote the painting's likeliness to the original image, and comprehensiveness, respectively. As Machado and Amaro generate pictures ranked highly by f_4 (see Figure 3a), they observe that the lines seem to be mostly circular. To encourage more varying line shapes they subtract $avg(avg(av))$ in f_6, resulting in lines with many direction changes, but overall circular velocity close to 0 (see Figure 3b).

(a) Two highest ranked paintings using f_4

(b) Two highest ranked paintings using f_6

Figure 3: Highest ranking ant colony paintings for fitness functions f_4 and f_6 as showcased in [4].

5 Evolutionary Figurative Images

This section outlines the framework by Correia et al. [3], which creates greyscale figurative images in four different categories: faces, lips, breasts and leaves. The work builds upon a previous paper by Machado et al. [6], in which only images of faces were created; furthermore, some of the authors have previously experimented with partly same tools and settings in [5, 9]. The framework contains two main modules: evolutionary engine and image classification. Next, the characteristics of evolutionary engine are discussed, which is followed by description of the image classification system. The section ends by reporting some of the interesting results obtained.

5.1 Genetic Programming Engine

The figurative image framework uses general purpose, expression based, genetic programming engine. The individuals of the population are rooted binary trees, where each internal node represents a function and terminal nodes consist of x, y and random constants. An example expression tree can be seen in Figure 4. The expression itself can be easily constructed, e.g., with a prefix notation, by visiting the nodes starting from the root and always visiting the left child first.

The x and y in the expression tree mark the location of the pixel in an image. This allows the image to be generated by evaluating the resulting expression for every x and y value combination, and using it as the darkness in the location (x, y). In previous works, also color images were generated by changing terminal nodes to

contain triplet values. However, the image classification system described in the next section handles only greyscale images; therefore, restricting the authors.

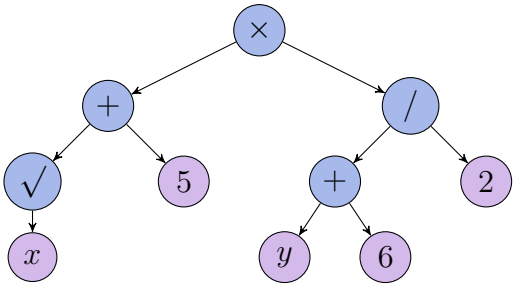


Figure 4: Example of an expression tree, which constructs an expression $\times$ (+ ($\sqrt{}$ x) 5) (/ (+ y 6) 2) in the prefix notation, or $(\sqrt{x} + 5) \times ((y + 6)/2)$ in the conventional notation.

Genetic operators mutate the tree structure and node contents. Crossover is accomplished by swapping random sub-trees of two individuals. Other mutations include sub-tree replacement within individual, and node insertion, deletion and mutation. Table 2 contains an overview of the GP engine parameters used in [3]. One prominent observation regarding the parameters is that also unary functions are used, so the tree is not necessarily full – when mutating unary function node into binary function, the system must also add a new terminal node.

Parameter	Setting
Population size	100
Number of generations	100
Crossover probability	0.8 (per individual)
Mutation probability	0.05 (per node)
Mutation operators	sub-tree swap, sub-tree replacement, node insertion, node deletion, node mutation
Initial maximum depth	5
Mutation max tree depth	3
Function set	+, −, ×, /, min, max, abs, neg, warp, sign, sqrt, pow, mdist, sin, cos, if

Table 2: Different parameters of the figurative image GP engine, see [5] for more setting details.

5.2 Image Classification

The image classification module uses Haar Cascade classifiers.[4] These classifiers generate a chain of successively more complex classifiers which use a set of Haar features. The Haar feature set represents different contrast change types in the image, which can be seen in Figure 5. The classifier is trained with positive and negative examples of the desired object category in order to generate the cascading structure, resulting in one "parent" classifier for each object category.

The images generated by the framework are afterwards positively recognised to have the object, if each successive simple classifier results in a positive output. However, as the general GP engine can not handle evaluation functions which return only 1 or 0 as output, the fitness of the resulting image in certain object category is calculated from the overall amount of the simple classifiers that the generated image passes. More formally, the fitness function f_C for image m, and object category C, is defined as

$$f_C(m) = \sum_{i}^{n_m} d(i) \cdot i + n_m \cdot 10,$$

where n_m is the number of chained classifiers image m has passed, and $d(i)$ holds the maximum difference between the threshold to pass the ith classifier and the value the image m acquired at the same classifier.

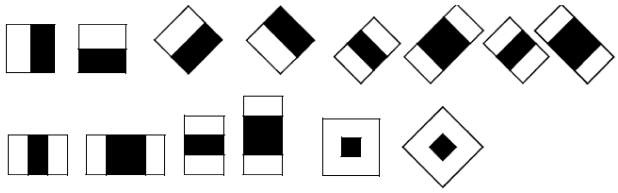


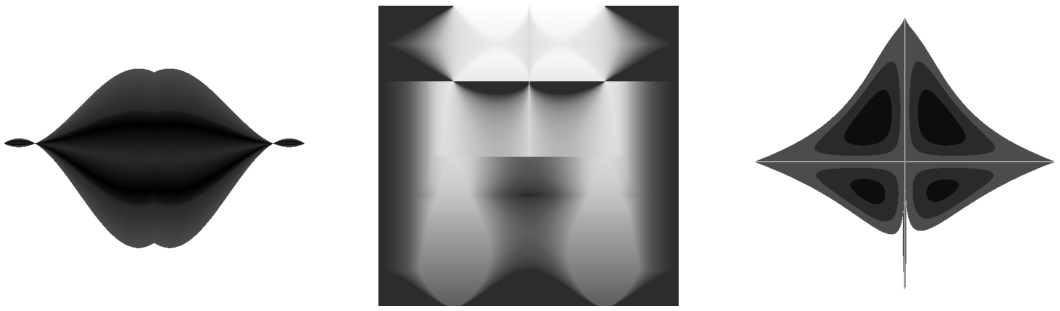
Figure 5: Set of Haar features used in the image classification

5.3 Results

Correia et al. experiment with their framework by generating sample images that are recognised to each category. They perform 30 independent runs for the evolutionary engine for each image category and obtain images that are succesfully classified to the used category in every run. However, not all of these images are observed as belonging to the category in question when viewed by a human. In Figure 6 can be

[4]The details of Haar Cascade classifiers are not in the scope of this report, see [12] for more detailed description.

seen examples of generated images with high fitness, which also reveal desired traits, for lips, face and leaf categories.



(a) Lips (b) Face (c) Leaf

Figure 6: Images showing desirable traits for lips, face, and leaf categories.

6 Evaluation

As Wiggins' model is purely mathematical construct, with only loose definitions of the purposes for different terms, it is impossible to have meaningful mappings from the system's implementation details to the model, without anchoring at least the desired conceptual space $\mathcal{C}$.

For ant colony paintings, the conceptual space will be defined as all the images that can be created in the *initial* boundaries of the evolutionary engine's parameters (and other hard coded features, such as initial placement of ants), given the input image. This definition is used to emphasize the fact that, although exploration is conducted, the restrictions for the conceptual space made by the evolutionary engine's allowed parameter value ranges do not change during the search. However, the paintings done with different ant colonies do exhibit visually different traits. Perhaps most notably, the changes in antennae vectors and weights allow interesting behaviours to emerge for the displacement vector $\Delta\vec{p}$ shown in Equation 1, which in turn is seen as different trail types in the paintings.

Figurative image framework suffers from another kind of disability, which is articulated by defining the conceptual space as all the different images that can be classified to certain object type using the allowed values of the GP engine, and the given object classifier. Although one could say the system is imaginative in the sense that it produces new objects defined by the restrictions of the classifier, it is still only able to create objects of types it has seen before. Importantly, it is fundamentally handicapped by only being able to produce one type of object at a time. However,

the search itself can be seen to be creative because it conforms to general genetic algorithm principles with mutation and evaluation criteria, which add surprisingness and appreciation, respectively.

7 Conclusions

Two different image creation systems were described and evaluated against Wiggins' model.

Machado and Pereira [7] describe an interactive painting tool using evolutionary engine that utilises ant colonies and their behaviour to create non-photorealistic image renderings. Each ant colony's genotype composes from general species traits, and each ant has few individual parameters that describe its current status. The population of the evolutionary engine is a set of ant colonies, each on its own copy of input image; the genetic operators are applied to the ant colony parameters and between the colonies. Machado and Amaro [4] continue the framework by automating the evolutionary engine by experimenting with different fitness functions for ant colonies.

Machado and Amaro conclude that it is possible to create reasonably simple fitness functions, based on general image and ant behaviour properties, which also produce interesting results. In essence, the procedure of the ant colony painting system produces just filters. After an ant colony creating visually pleasing artefacts is found, it can be applied to any input image with, while not deterministic, at least consistent effects. To this extent, the creativity of the system is in the search for the new rendering types, or styles as mentioned by Colton. The system does not paint scenes from the imagination, as it is strictly restricted in producing renderings from the input image. To allow more "imagination" for the system, the parameter values could be, e.g., shifted according to high fitness individuals. Also, a process for altering the fitness function during the execution of the search could be deployed.

Correia et al. [3] use genetic programming and image classification system to create figurative images in four different categories. Their paper leans heavily on the earlier work of Machado et al. [6], in which only one type of objects were created. The main contribution of the Correia et al. is the generalisation of the types of objects created by showing that a generic classifier learning process can be used, without the need of handbuilding a new classifier for each object category.

Correia et al. show, that the objects created can be classified effectively by the same classifiers that are used to learn object categories. However, they also admit that the system creates examples that are classified to a certain category with high

certainty, but are not recognised to belong into it by a human viewer. Unfortunately, being forced to handpick and preprocess the learning set images, and the ability to only produce objects of one category at one run of evolutionary engine, highly limits the artistic capabilities of the system. Enhancing the system's autonomy in object creation by using multi-label classification chains could produce interesting category blending techniques – if correctly applied.

Overall, it seems that the systems described in this report exhibit some creativity in their behaviour, but because of their simplified realities of being only mass producers of images, they lack true conceptual blending and cross-domain analogy techniques. This seems like an interesting direction for the future work: how image creation systems could behave as a society?

References

[1] Boden, M.A.: *The creative mind: myths & mechanisms*. Basic Books, 1990, ISBN 9780465014521. http://books.google.fi/books?id=BEbuAAAAMAAJ.

[2] Colton, Simon: *The painting fool: Stories from building an automated painter*. In McCormack, Jon and Mark d'Inverno [10], chapter 1, pages 3–38, ISBN 978-3-642-31726-2 (PRINT) 978-3-642-31727-9 (ONLINE).

[3] Correia, João, Machado, Penousal, Romero, Juan, and Carballal, Adrián: *Evolving figurative images using expression-based evolutionary art*. In *Proceedings of the fourth International Conference on Computational Creativity (ICCC)*, pages 24–31, 2013.

[4] Machado, Penousal and Amaro, Hugo: *Fitness functions for ant colony paintings*. In *Proceedings of the fourth International Conference on Computational Creativity (ICCC)*, pages 32–39, 2013.

[5] Machado, Penousal and Cardoso, Amílcar: *All the truth about nevar*. Appl. Intell., 16(2):101–118, 2002. http://dblp.uni-trier.de/db/journals/apin/apin16.html#MachadoC02.

[6] Machado, Penousal, Correia, João, and Romero, Juan: *Expression-based evolution of faces*. In Machado, Penousal, Romero, Juan, and Carballal, Adrián (editors): *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, volume 7247 of *Lecture Notes in Computer Science*, pages 187–198. Springer Berlin Heidelberg, 2012, ISBN 978-3-642-29141-8. http://dx.doi.org/10.1007/978-3-642-29142-5_17.

[7] Machado, Penousal and Pereira, Luís: *Photogrowth: non-photorealistic renderings through ant paintings*. In Soule, Terence and Moore, Jason H. (editors): *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 233–240. ACM, 2012.

[8] Machado, Penousal, Romero, Juan, and Manaris, Bill: *Experiments in computational aesthetics: An iterative approach to stylistic change in evolutionary art*. In Romero, Juan and Machado, Penousal (editors): *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pages 381–415. Springer Berlin Heidelberg, 2007.

[9] Machado, Penousal, Romero, Juan, and Manaris, Bill: *Experiments in computational aesthetics*. In Romero, Juan and Machado, Penousal (editors): *The Art of Artificial Evolution*, Natural Computing Series, pages 381–415. Springer Berlin Heidelberg, 2008, ISBN 978-3-540-72876-4. `http://dx.doi.org/10.1007/978-3-540-72877-1_18`.

[10] McCormack, Jon and d'Inverno, Mark (editors): *Computers and Creativity*. Springer, Berlin; Heidelberg, 2012, ISBN 978-3-642-31726-2 (PRINT) 978-3-642-31727-9 (ONLINE).

[11] Mitchell, Melanie: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, USA, 1998, ISBN 0262631857.

[12] Viola, P. and Jones, M.: *Rapid object detection using a boosted cascade of simple features*. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[13] Wiggins, Geraint A.: *A preliminary framework for description, analysis and comparison of creative systems*. Know.-Based Syst., 19(7):449–458, November 2006, ISSN 0950-7051. `http://dx.doi.org/10.1016/j.knosys.2006.04.009`.