Computational Creativity Theory: The FACE and IDEA Descriptive Models

Simon Colton[1], **John Charnley**[1] **and Alison Pease**[2]

[1] Computational Creativity Group, Department of Computing, Imperial College, London, UK. ccg.doc.ic.ac.uk

[2] School of Informatics, University of Edinburgh, UK.

Abstract

We introduce computational creativity theory (CCT) as an analogue in computational creativity research to computational learning theory in machine learning. In its current draft, CCT comprises the FACE descriptive model of creative acts as tuples of generative acts, and the IDEA descriptive model of the impact such creative acts may have. To introduce these, we simplify various assumptions about software development, background material given to software, how creative acts are performed by computer, and how audiences consume the results. We use the two descriptive models to perform two comparisons studies, firstly for mathematical discovery software, and secondly for visual art generating programs. We conclude by discussing possible additions, improvements and refinements to CCT.

Introduction

Machine learning is a very well established research area, and has possibly had the greatest impact of all AI subfields, with successful applications to classification and prediction tasks in hundreds of areas of discourse. Machine learning researchers asked: "What does it mean to say that a computer has *learned* something?" In computational creativity research, we have similarly asked: "What does it mean to say that a computer has *created* something?" It is therefore sensible to look at how machine learning developed to suggest future directions for computational creativity research. Human learning involves many behaviours, including memorisation, comprehension, abstraction and generalisation. While textbooks like (Mitchell 1997) use general terms such as "learning how to do something by observing it being done", in reality, machine learning researchers have largely studied how to induce a concept definition (or set of rules) to fit data which has been given in advance. The induced definition may be a Prolog program, a decision tree, an artificial neural network, etc., and can be used to classify new examples with respect to the concept. With careful manipulation, many intelligent tasks can be stated as such concept-induction problems. In particular, rules for predicting attributes of unseen objects, such as the toxicity of a particular drug, can be learned and put to great practical use.

The focus on concept induction enabled a formalisation of acts of learning to be explored within the field of *computational learning theory* (CLT). In (Angluin 1992), Dana Angluin states that the goals of CLT are to:

> *Give a rigorous, computationally detailed and plausible account of how learning can be done.*

Each word in this statement has been carefully chosen. In particular, by requiring *computational detail*, Angluin highlights that CLT discusses actual algorithms which learn. By requiring *rigour*, he stresses that the purpose of CLT is to state and prove theorems about the nature of those learning algorithms. Such theorems deal with learnability within the PAC model (Valiant 1984), i.e., which types of concepts can be induced by which methods, and with efficiency, i.e., the extent to which certain methods must search a space. By mentioning the *plausibility* of the account offered, Angluin grounds CLT to describing machine learning in reality, i.e., with reasonable computational resources, background information and interaction requirements. Finally, by softening the account to one which describes how *learning can be done*, Angluin emphasises that CLT does not aim to wholly encompass approaches to computer learning, but rather offers a particular formalism which researchers can use to assess progress. The benefits of CLT have been demonstrated via theoretical results being turned into practical application, most notably the introduction of ensemble learning methods such as boosting (Schapire 1990), whereby powerful predictors are assembled from multiple weak ones.

We introduce here the first draft of a formalism called *computational creativity theory* (CCT). In the long term, we aim for CCT to provide a rigorous, computationally detailed and plausible account of how creation can be done. Our first step has been to provide two plausible descriptive models which can be used to describe the processing and the output of software designed to exhibit creative behaviour. We restrict CCT entirely to describing software, and we make no claims about its value for describing human behaviour. We start at the base level of creativity: the *creative act*, whereby something – however trivial – has been created by computer, and we introduce the FACE descriptive model for describing such creative acts. We further introduce the IDEA descriptive model which captures notions related to the impact of such creative acts.

In the spirit of Angluin's mission statement above, we do not aim for CCT to describe every way in which software

could be creative, but rather *a plausible way* in which creation by software could occur. We therefore make some simplifying assumptions that may exclude some types of creative acts. Having said this, we believe the models are sufficiently flexible to describe in general terms the output and processing of the artefact generation systems that have been discussed in computational creativity circles in recent years. How the FACE and IDEA models are employed to compare and contrast creative acts is in part generic, and in part domain-specific. To highlight the potential for describing creative acts by computer and their impact, in the penultimate section, we use the models in comparison studies of generative mathematics and generative art systems. We conclude by highlighting areas where computational creativity theory could be expanded or refined.

Related Work

The formalisms presented below draw heavily from existing formalisms of creative systems. While CCT does not yet capture all aspects of these existing formalisms, we have tried to include various notions which are commonly agreed upon to be important in computational creativity research. In particular, CCT has been motivated by notions of exploratory and transformational search described in (Boden 2003) and (Wiggins 2006). We have also tried to capture notions of meta-level reasoning, as described in (Buchanan 2001) and (Colton 2009), in addition to the usage of input/output for assessment of creative activity put forward in (Ritchie 2007). We have similarly tried to capture the important notion of surprise described in (Macedo and Cardoso 2002), and notions of how audience expectation and the perception of creativity must be handled, as described in (Colton 2008). We have also drawn from the history of the field of computational creativity, as given in (Cardoso, Veale, and Wiggins 2009). Our motivation for the descriptive models also came from writings on human creativity within the philosophy and psychological literature, such as (Sloman 1978), who sets out criteria which can be used to judge theories; (Thagard 1993), where criteria for evaluating explanatory theories are also set out; and (Uzzi and Spiro 2005) who describe societal influences on creativity. Due to lack of space here, we have placed CCT into such contexts in a sister paper (Pease and Colton 2011a).

The FACE Descriptive Model of Creative Acts Performed by Software

We aim to capture some basic types of creative act performed by computer. It is clear that such acts can occur at the ground level, whereby generative systems produce new artefacts such as theorems, pictures, compositions, poems, etc. However, it is also clear that creative acts can occur at the process level, where new ways to generate and assess artefacts are invented. We define a *creative act* as a non-empty tuple of generative acts. Each tuple contains exactly zero or one instance of eight types of individual generative act. The different types of generative act are denoted by the following letters with g or p superscripts, representing generative acts which produce:

- E^g: an expression of a concept
- E^p: a method for generating expressions of a concept
- C^g: a concept
- C^p: a method for generating concepts
- A^g: an aesthetic measure
- A^p: a method for generating aesthetic measures
- F^g: an item of framing information
- F^p: a method for generating framing information

By the word *concept*, we mean an executable program (or something which can be interpreted as such), which is capable of taking input and producing output. By the phrase *expression of a concept*, we mean an instance of an (input, output) pair produced when the concept's program is executed. By the term *aesthetic measure*, we mean a function which takes as input a (concept, expression) pair – one of which can possibly be null – and outputs a real value between 0 and infinity. By an item of *framing information*, we mean a piece of natural language text that is comprehensible by people, which refers to a non-empty subset of generative acts in the tuple. Such framing information may be domain-specific, and adds value to the generative acts, possibly by (a) putting them into some cultural or historical context, or (b) describing the processes underlying the generative acts, or (c) providing calculations about the concepts/expressions with respect to the aesthetic measures, or (d) obfuscating the creative process and/or the output produced, in order to increase the amount of interpretation required by audience members. By allowing exactly one or zero of each type of generative act, the above definition of a creative act includes singular generative acts. Such acts include those performed by machine learning systems which generate a single classifying concept: $\langle C^g \rangle$, and those performed by constraint solvers, where a single expression (the solution) of a concept (the constraint satisfaction problem) is produced: $\langle E^g \rangle$.

In writing $\langle F^g, A^g, C^g, E^g \rangle$, we denote a creative act comprising a 4-tuple of generative acts. This explicitly implies that E^g produced an expression of the concept produced by C^g; that the output from the pair of generative acts (C^g, E^g) can be taken as input by the aesthetic measure generated by A^g; and that F^g generated an item of framing information which further explains or justifies the whole creative act. A creative act can be probed for information about both the generative processes it contains and the output from these processes. To simplify matters, we use lower-case notation to denote the output from the generative acts. Hence we might describe the creative act $\langle F^g, A^g, C^g, E^g \rangle$ as producing framing information f^g which justifies the creation of aesthetic measure a^g by which the concept/expression pair (c^g, e^g) can be assessed – noting that e^g is an expression of c^g, and both were generated during this creative act. In order to describe creative acts in the context of software development and deployment, where a programmer, user and/or an audience member may contribute to a creative act, we use a bar notation to highlight generative acts undertaken by a third party. For instance, the tuple $\langle \overline{A^g}, \overline{C^g}, E^g \rangle$ might represent the creative act $\langle E^g \rangle$ performed by the software whereby an expression of user-given concept $\overline{c^g}$ was generated and assessed by user-given aesthetic measure $\overline{a^g}$.

Note that creative acts need only contain a single trivial generative act, e.g., $\langle E^g \rangle$ might represent multiplying two numbers together, which we include as a creative act. Also, in tuples which include matching generative acts with both g and p superscripts (indicating ground and process-level acts respectively), the implication is that the process level generative act produced a method employed by the ground level generative act. For instance, we might use $\langle C^p, C^g, E^g \rangle$ to denote a creative act whereby a software system undertook a process level generative act, C^p, which invented a new method for generating concepts, c^p, and then used that method in a concept-generating act, C^g, which produced a concept, c^g, which was in turn used in an expression generating act, E^g, to produce the e^g expression of c^g.

The FACE model could be used in a *quantitative way*, with sheer volume of creative acts used to compare creative systems. It might be more informative, however, to use it in a *cumulative way*, where a creative act is seen as strictly more creative than another if strictly more types of generative acts occurred. For example, if creative act CA_1, invents a concept and an expression of that concept, it should be seen as less creative than creative act CA_2 which also invents an aesthetic against which the same concept and expression could be assessed. Here, using the $>$ and $<$ symbols to represent 'more creative than' and 'less creative than', we could write: $CA_1 = \langle C^g, E^g \rangle < CA_2 = \langle A^g, C^g, E^g \rangle$. The FACE model could also be used in a *comparative way*, where undertaking one type of generative act is seen as more creative than another. Hence, some orderings could be agreed upon, like: $\langle C^p \rangle > \langle A^g \rangle > \langle C^g \rangle > \langle E^g \rangle$. Moreover, it could be used in a *process-based way*, with individual generative acts compared, e.g., random generation might be seen as less creative than an inductive method, etc.

Finally, the FACE model could be used in a *qualitative way*, where the values from aesthetic functions are used to compare creative acts. Suppose that we want to compare sessions during which an existing aesthetic measure $\overline{a^g}$ is given, and the software successively produces concept/expression pairs $(c_1^g, e_1^g), \ldots, (c_n^g, e_n^g)$, which can be assessed by $\overline{a^g}$. Suppose further that a threshold t for minimum acceptable aesthetic level is given, below which concepts/expressions will be deemed as too low quality. The following measures could be used to assess software S during such a session of creative acts:

$$average(S) = \frac{1}{n} \sum_{i=1}^{n} \overline{a^g}(c_i^g, e_i^g)$$
$$best\_ever(S) = max_{i=1}^{n} (\overline{a^g}(c_i^g, e_i^g))$$
$$worst\_ever(S) = min_{i=1}^{n} (\overline{a^g}(c_i^g, e_i^g))$$
$$precision(S) = \frac{1}{n} |\{(c_i^g, e_i^g) : 1 \leq i \leq n \wedge \overline{a^g}(c_i^g, e_i^g) > t\}|$$
$$reliability(S) = best\_ever(S) - worst\_ever(S)$$

Measures similar to the first three are regularly used in the development and usage of creative systems. Precision might be a valuable measure for comparing systems in domains where the yield of concepts/expressions of an acceptably high standard is low. The range between the best and worst outputs of a session, as assessed by the reliability measure, might be used while developing software, i.e., in order to increase the trust the programmer/user has in the ability of the software to deliver consistent results.

The IDEA Descriptive Model for the Impact of Creative Acts Performed by Software

In problem solving applications of AI techniques, such as theorem proving or constraint solving, an aesthetic measure is usually given which effectively measures the value of the results of the creative acts (the proposed problem *solutions*) in terms of how well they solve the problem. Alternatively, when simulating human intelligence is the aim, Turing-style aesthetics may be used, where the measure of the creative output (e.g., human-computer dialogues) is in terms of how much audiences are fooled into projecting humanity onto computer output. In such applications, there are fairly concrete notions of *right and wrong*, and hence the value of software for solving/simulation can be objectively assessed. However, in more creative applications of AI techniques, the invention of aesthetic measures may be part of the task, hence there is no a-priori notion of right or wrong (as discussed at length in (Pease and Colton 2011)). For this reason, we replace the usual notion of *value of solutions* in the AI problem solving paradigm with the notion of *impact of creations* in the AI artefact generation paradigm.

Software is not developed in a vacuum, so we must factor out the input of the programmer/user/audience when we assess the impact of the creative acts performed by software. To do so, we assume an (I)terative (D)evelopment-(E)xecution-(A)ppreciation cycle within which software is engineered and its behaviour is exposed to an audience. For this IDEA model, we wish to capture the notion that the fine-tuning of software has an effect on the impact of its creative acts, as discussed in (Colton, Pease, and Ritchie 2001) and (Ritchie 2007). We also wish to capture the notion that the novelty of the results of a creative act can affect people's appreciation of those results (Macedo and Cardoso 2002). In particular, positive appreciation tends to peak as novelty increases, but as the results of a creative act become too novel, it becomes difficult to put them into context, and overall appreciation of them drops (Wundt 1874). We make three simplifying assumptions about the development/deployment of a software system S, as follows:

• *An ideal development process*. S will eventually perform creative acts such as those describable by the FACE model. It is programmed by someone with knowledge/experience of the kinds of creative acts that S will perform, as described by the *ideal background knowledge information* assumptions below. S is designed to surprise the programmer, by performing creative acts and producing outputs which were not predicted in advance. During the development stage, the programmer repeatedly executes S, assesses its performance, and makes improvements accordingly. Once fully programmed, S is executed by either the programmer and/or a third party user and performs creative acts, the results of which are assessed by an *ideal audience*, as described below.

• *Ideal background knowledge information*. When developing S, the programmer is aware of three sets of knowledge, namely α: all the relevant knowledge known to the world; β: the subset of α given explicitly or implicitly to S at design and/or run time; and κ: the knowledge produced by S (some of which may overlap with α). Each knowledge set is

represented as a set of FACE creative act tuples and the outputs from the tuples. In addition, there is available a distance measure, denoted $d(t_1, t_2)$, which can take into account the processes behind and the output from two creative act tuples t_1 and t_2, and estimate how similar they are.

• *An ideal audience*. Judgement/appreciation of an act of creativity, A, performed by S is undertaken by an audience of idealised people, each able to give two ratings for A. For audience member m, they first give a *well-being rating*, $wb_m(A)$, between -1 and +1, with -1 indicating a strong dislike for A, +1 indicating a strong liking for A, and 0 indicating no strong opinion. Secondly, they give a *cognitive effort rating*, $ce_m(A)$ between 0 and 1, with 0 indicating that they were prepared to spend no time attempting to understand A, and 1 indicating that they were prepared to spend a long time attempting to understand A. With these ratings, we avoid circularity by not expecting audience members to evaluate creativity directly, and we attempt to capture temporal aspects of evaluation, as introduced below.

It is likely that less creativity will be attributed to the software, if the creative acts it undertakes are too similar to known ones, especially if known to the programmer. If less creativity is attributed to the software, then it is likely to have less impact. With this in mind, we suppose that there is a threshold, l, on the distance measure d such that any creative act is deemed too similar to a known creative act (possibly undertaken by the programmer) if the distance between the two creative acts is less than l. For instance, if the programmer wrote a production rule which was able to create a new concept from a specific given concept, then we might argue that the creative act of inventing this production rule, $\overline{A} = \langle \overline{C^p} \rangle$, is too similar to the act of applying the rule to generate a concept, $A = \langle C^g \rangle$, i.e., $d(\overline{A}, A) < l$. Note here that we have compared two distinct types of creative acts, one producing a method for generating concepts, and one actually producing concepts. We similarly suppose there is an upper threshold, u, for the distance between a known creative act and one undertaken by the software, such that if the distance between the two creative acts is greater than u, then there is no context within which they can be compared. Using such upper and lower bound thresholds, we can suggest the following engineering stages for software S:

• *Developmental stage:* where all the creative acts undertaken by S are based on inspiring examples (c.f. (Ritchie 2007)), i.e., $\forall\, K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K, B) = 0)$.

• *Fine tuned stage:* where the creative acts performed by S are abstracted away from inspiring examples, but are still too close to have an impact as novel inventions, i.e., $\forall\, K \in \kappa, (\exists B \in \beta \text{ s.t. } d(K, B) < l)$.

• *Re-invention stage:* where S performs creative acts similar to ones which are known, but which were not explicitly provided by the programmer, i.e., $\exists\, K \in \kappa$ s.t. $(\exists\, A \in \alpha \text{ s.t. } (d(K, A) < l \wedge A \notin \beta))$.

• *Discovery stage:* where S performs creative acts sufficiently dissimilar to known ones to have an impact due to novelty, but sufficiently similar to be assessed within current contexts, i.e., $\exists\, K \in \kappa$ s.t. $((\nexists\, A \in \alpha \text{ s.t. } d(K, A) < l) \wedge (\exists A' \in \alpha \text{ s.t. } d(K, A') < u))$.

• *Disruption stage:* where S performs some creative acts which are too dissimilar to those known to the world to be assessed in current contexts, hence new contexts have to be invented, i.e., $\exists\, K \in \kappa$ s.t. $(\nexists\, A \in \alpha \text{ s.t. } d(K, A) < u)$.

• *Disorientation stage:* where all the creative acts performed by S are too dissimilar to known ones that there is no context within which to judge any of its activities, i.e., $\forall\, K \in \kappa, (\nexists\, A \in \alpha \text{ s.t. } d(K, A) < u)$.

To use these stages of software development in an impact model, we note that it is unlikely that software in any stage other than the discovery or disruption stage will have much impact. This is because in the earlier stages, S acts too much like an avatar for the programmer, while in the disorientation stage, its behaviour is too strange for it to be taken seriously.

Turning next to the possibilities for using the effect on an ideal audience in judging impact, we assume an audience of n idealised people passing judgement on a creative act A, whereby full disclosure of the act itself and the results of the act are provided to each audience member. We denote the mean value of the well-being rating over the n people as $m(A)$, and we propose the following measures which could be used in impact assessment exercises:

$$dis(A) = disgust(A) = \tfrac{1}{2n} \sum_{i=1}^{n} (1 - wb_i(A))$$
$$div(A) = divisiveness(A) = \tfrac{1}{n} \sum_{i=1}^{n} |wb_i(A) - m(A)|$$
$$ind(A) = indifference(A) = 1 - \tfrac{1}{n} \sum_{i=1}^{n} |wb_i(A)|$$
$$pop(A) = popularity(A) = \tfrac{1}{2n} \sum_{i=1}^{n} (1 + wb_i(A))$$
$$prov(A) = provocation(A) = \tfrac{1}{n} \sum_{i=1}^{n} (ce_i(A))$$

Each measure returns a value between 0 and 1. We see that *disgust* and *popularity* measure the average distance from the extremes of the well-being measure, i.e., how much people dislike and like the creative act (and its output) on average, respectively. Moreover, the *divisiveness* measure is maximised when half the audience thoroughly dislike A and the other half thoroughly like A, which captures some notion of divisiveness; *indifference* is maximised when people have no strong well-being opinion about A; and *provocation* records how much thought the audience members are prepared to put into understanding A. By compounding the provocation measure with the others, we can attempt to capture some kinds of impact that creative acts might have:

$$acquired\_taste(A) = (pop(A) + prov(A))/2$$
$$instant\_appeal(A) = (1 + pop(A) - prov(A))/2$$
$$opinion\_splitting(A) = (1 + div(A) - prov(A))/2$$
$$opinion\_forming(A) = (div(A) + prov(A))/2$$
$$shock(A) = (1 + dis(A) - prov(A))/2$$
$$subversion(A) = (dis(A) + prov(A))/2$$
$$triviality(A) = (1 + ind(a) - prov(A))/2$$

Again, each of these measures returns a value between 0 and 1. It is possible to argue that – with the exception of triviality – if A reaches a score towards 1 for any of these measures, it has had some impact. In particular, there are instant impacts, modeled by the *instant_appeal*, *shock* and *opinion_splitting* measures, which increase as the provocation level decreases. There are also more considered impacts, modeled by the *acquired_taste*, *opinion_forming* and *subversion* measures, which increase as the provocation level increases. Finally, *triviality* measures the situation whereby people come to an instant indifferent opinion about A.

Comparison Studies

Generative mathematics systems include automated reasoning systems such as model generators and theorem provers, which perform singular creative acts of the form $\langle E^g \rangle$ and $\langle C^g \rangle$ respectively (if we note that a proof can be interpreted as an executable program, for instance by proof checkers). The most studied mathematics software from a computational creativity perspective have been *theory formation systems*, which perform concept formation, conjecture making, proving/disproving, etc. The first these was AM (Lenat 1982), which performed theory formation in set theory and number theory. Later, the notions of production rules, heuristics and concepts, which were conflated in AM, were identified, extracted and clarified in the theory formation performed by the HR system (Colton 2002).

AM and HR both perform creative acts of the form $\langle \overline{A^g}, C^g, E^g \rangle$, where concepts/conjectures are invented/discovered and expressions calculated, with the definitions/expressions being used to determine value against user given aesthetics (known as measures of interestingness, which drive a heuristic search). In addition, HR performs theorem proving acts $\langle C^g \rangle$ and model generation acts $\langle E^g \rangle$, using third party systems. Under the FACE model, using a cumulative approach, HR and AM are equally creative as both generate concepts and expressions. Using a process-based approach, HR performs more creative acts (proving and disproving), hence is more creative. Using a qualitative approach, the precision of AM was higher than HR, i.e., in theory formation sessions, AM produced higher yields of interesting concepts/conjectures than HR. However, AM has been heavily criticised by a number of different authors. In particular, as discussed in (Colton, Pease, and Ritchie 2001), with more production rules than concepts formed in any session, AM was fine-tuned to produce certain results. This almost certainly accounts for the high precision AM enjoyed. Moreover, while Lenat claimed the invention of novel, interesting mathematical concepts, this was not the case. In contrast, HR produces thousands of concepts in a session using only a handful of production rules, and has contributed to the mathematical literature in number theory and algebraic domains. Under the IDEA model, it seems fair to say that AM never reached the discovery stage, whereas HR has.

In (Colton 2001), HR was used to form theories about how concepts are formed and assessed, using previously formed theories as background knowledge. The kinds of creative acts it undertook were of the form $\langle C^p, C^g, E^g \rangle$ and $\langle A^g, C^g, E^g \rangle$, which resulted in a new measure of interestingness being added to HR's core functionality. Using the cumulative approach within the FACE model, this version of HR was more creative than the standard one. Also, in (Pease 2007), Lakatos-style concept refinement techniques were added to the theory formation routine in HR, enabling more sophisticated concept formation which would be seen as more creative under a process-based FACE analysis. This led to the TM system described in (Colton and Pease 2005), which produced provably-true variations of given non-theorems. This therefore creatively extended model generation from performing a single $\langle E^g \rangle$ creative act to performing multiple $\langle C^g, E^g \rangle$ acts.

Generative art systems include those which turn mathematical equations such as fractals into abstract art, and those which apply image filters to a given digital photograph, both of which perform singular creative acts of the form $\langle E^g \rangle$. Most studied within computational creativity research are automated painter systems and evolutionary art software. The AARON system (McCorduck 1991) is the most well-known automated painter system, which has been used by Harold Cohen to produce valued artworks for around 40 years. In recent years, The Painting Fool system (www.thepaintingfool.com) has also been developed to automate aspects of the painting process. Both systems perform creative acts of the form $\langle C^g, E^g \rangle$, with concept formation being the invention of scenes to paint, and concept expressions being the rendering of those scenes.

In (Krzeczkowska et al. 2010), The Painting Fool was given the ability to turn news articles into collages via text extraction and image retrieval. It is possible to argue that this is a creative extension, because it enabled The Painting Fool to perform creative acts of the form $\langle A^g, C^g, E^g \rangle$, with a local aesthetic – namely for the collage to properly illustrate the theme of the news article – being generated alongside a concept (a spatial organisation of retrieved images) and an expression (painting) of that concept. Note that The Painting Fool does not actually apply the aesthetic measure to the collages it generates. In contrast, in (Colton 2008a), The Painting Fool explicitly generated mathematical fitness functions as aesthetic measures, using the HR system, and applied these measures to evolve scenes (cityscapes and flower arrangements), which were rendered to produce paintings. Hence, the results from the creative acts here could be described as: $\langle a^g, c^g, e^g \rangle$ = $\langle$ fitness_function, scene, render $\rangle$.

The NEvAr evolutionary art system (Machado and Cardoso 2000) can be driven by a person expressing their aesthetics considerations while choosing between generated images. However, NEvAr can also appeal to programmed-in measures of aesthetic value based on information theoretic routines, amongst others. Using such measures, it can drive its own search for images. We can compare NEvAr to the Avera evolutionary system described in (Hull and Colton 2007), which has no such aesthetic measures built in, hence is entirely driven by user choices. Under the FACE model, NEvAr and Avera both perform creative acts of the form $\langle C^g, E^g \rangle$, with the concepts they invent being crossed-over and mutated genomes which are expressed as images. The more autonomous nature of NEvAr means that it should be assessed as more creative under a progress-based assessment, and its usage of aesthetic measures will increase its precision, hence it should be seen as more creative under a qualitative assessment.

Neither of the above comparison studies is particularly comprehensive or acute. In fact, we have not used many of the tools available from the FACE or IDEA models. However, it is interesting to note that the systems which are usually not considered within the computational creativity remit perform singular generative acts, whereas more creative system produce larger tuples of size two or three. In future, we will produce longer and more precise accounts of creative software in these and other application domains.

Conclusions and Future Work

We have presented the first draft of computational creativity theory (CCT), which comprises two descriptive models: (i) the FACE model for describing creative acts performed by software in terms of tuples of generative acts, and (ii) the IDEA model of how such creative acts can have an impact. We showed how these models could be used to compare both mathematical invention software and visual art producing programs. Our main contribution has been to introduce a formal notion of the creative act, and to make this the centre of the theory. This has allowed us to generalise various long held practices in computational creativity research. For instance, rather than comparing just the output of systems, we propose to compare entire creative acts (which include details of the output *and* the processes). Rather than assessing the value of artefacts produced with respect to given aesthetic measures, we propose to assess the impact of creative acts, where aesthetic measures may be invented. Rather than describing/assessing individual generative acts, we propose to study tuples of generative acts comprising creative acts.

There are numerous extensions and refinements to CCT which we plan for future drafts. For instance, the IDEA model doesn't yet capture how creative acts can have impact by changing people's opinions, for instance by introducing them to new aesthetics. It also doesn't model how people's opinions change through group discussion, and it would be a more powerful tool if group dynamics and causal elements were included. We are also planning a third descriptive model which covers the notion of obfuscation, i.e., ways in which software could actively disguise its creative acts, to manage how much audience members must interpret them. We also plan to add more formal detail to the notion of framing information; the process-based way of comparing creative acts; and the way in which the similarity of creative acts can be estimated. By being inclusive about creative acts, we can say that most AI software performs such acts (but may not be assessed as particularly creative by the FACE or IDEA models). Hence, we can build on formalisms for other AI subfields. In particular, we could consider extending computational learning theory (CLT) with more aspects of the creative process. The two major aesthetics in CLT are predictive accuracy of classifiers and their parsimony with respect to certain algorithmic complexity measures. CLT could therefore be generalised to using arbitrary aesthetic measures, and further extended to include the invention of such aesthetics within the machine learning method.

It is our hope that computational creativity theory will be adopted for practical purposes when building software, i.e., to demonstrate progress from one version of a creative system to another, and to compare and contrast different software systems for similar creative purposes. In the long term, we hope that CCT could underpin *predictive* models for estimating in advance how creative a software system will be.

Acknowledgements

We wish to thank the anonymous reviewers for their helpful and encouraging comments, and members of the computational creativity steering group for their valuable input.

References

Angluin, D. 1992. Computational learning theory: Survey and selected bibliography. *Proc. 24th ACM symp. on Theory of Comp.*.

Boden, M. 2003. *The Creative Mind: Myths and Mechanisms (second edition)*. Routledge.

Buchanan, B. 2001. Creativity at the meta-level. *AI Mag.* 22(3).

Cardoso, A.; Veale, T.; and Wiggins, G. 2009. Converging on the divergent: The history (and future) of the international joint workshops in computational creativity. *AI Magazine* 30(3).

Colton, S., and Pease, A. 2005. The TM system for repairing non-theorems. *ENTCS*, 125(3).

Colton, S.; Pease, A.; and Ritchie, G. 2001. The effect of input knowledge on creativity. *Proc. ICCBR'01 Wshop on Creat. Syst.*.

Colton, S. 2001. Experiments in meta-theory formation. In *Proc. of the AISB'01 Symposium on AI and Creativity in Arts and Science*.

Colton, S. 2002. *Auto. Theory Formation in Pure Maths*. Springer.

Colton, S. 2008a. Automatic invention of fitness functions, with application to scene generation. In *Proc. of the EvoMusArt Wshop*.

Colton, S. 2008. Creativity vs the perception of creativity in computational systems. *Proc. AAAI Spring Symp. on Creative Systems*.

Colton, S. 2009. Seven catchy phrases for computational creativity research. In *Proceedings of the Dagstuhl Seminar: Computational Creativity: An Interdisciplinary Approach*.

Hull, M., and Colton, S. 2007. Towards a general framework for program generation in creative domains. In *Proceedings of the 4th International Joint Workshop on Computational Creativity*.

Krzeczkowska, A.; El-Hage, J.; Colton, S.; and Clark, S. 2010. Automated collage generation - with intent. In *Proceedings of the 1st International Conference on Computational Creativity*.

Lenat, D. 1982. AM: Discovery in mathematics as heuristic search. In *Knowledge-Based Systems in AI*. McGraw-Hill.

Macedo, L., and Cardoso, A. 2002. Assessing creativity: the importance of unexpected novelty. *Proc. ECAI Wshop on Creat. Syst.*.

Machado, P., and Cardoso, A. 2000. NEvAr – the assessment of an evolutionary art tool. In *Proc. of the AISB Symp. on Creative and Cultural Aspects and Applications of AI and Cognitive Science*.

McCorduck, P. 1991. *AARON's Code: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen*. W.H. Freeman and Co.

Mitchell, T. 1997. *Machine Learning*. McGraw Hill.

Pease, A. 2007. *A Computational Model of Lakatos-style Reasoning*. Ph.D., School of Informatics, University of Edinburgh.

Pease, A., and Colton, S. 2011a. CCT: Inspirations behind the FACE and IDEA models In *Proc. of the Int. Conf. on Comp. Creat.*

Pease, A., and Colton, S. 2011. On impact and evaluation in computational creativity In *Proc. of the AISB symp. on AI and Philos*.

Ritchie, G. 2007. Some empirical criteria for attributing creativity to a computer program. *Minds and Machines* 17.

Schapire, R. 1990. Strength of weak learnability. *Mach. Learn.* 5.

Sloman, A. 1978. *The Computer Revolution in Philosophy*. The Harvester Press.

Thagard, P. 1993. *Comp. Philosophy of Science*. MIT Press.

Uzzi, B., and Spiro, J. 2005. Collaboration and creativity: The small world problem. *Am. Journal of Sociology* 111(2).

Valiant, L. 1984. A theory of the learnable. *Comm. ACM* 27(11).

Wiggins, G. 2006. Searching for computational creativity. *New Generation Computing* 24(3).

Wundt, W. 1874. *Grundzuge der Phvsiologischen Psychologie*. Engelmann.