Ten Steps to Make a Perfect Creative Evolutionary Design System

Peter J. Bentley
Department of Computer Science
University College London
Gower Street
London WC1E 6BT
P.Bentley@cs.ucl.ac.uk
http://www.cs.ucl.ac.uk/staff/P.Bentley/

Una-May O'Reilly
Artificial Intelligence Lab
Massachusetts Institute of Technology
Cambridge
MA, 02139
unamay@ai.mit.edu
http://www.ai.mit.edu/people/unamay

Abstract

A perfect creative evolutionary design system is impossible to achieve, but in this position paper we discuss 10 steps that might bring us a little closer to this dream. These important problems and requirements have been identified as a result of both authors' experiences on a number of projects in this area. While our solutions may not solve all of the problems, they illustrate what we regard as the current state of the art in creative evolutionary design.

1 INTRODUCTION

Sometimes it's good to let the imagination run riot. To let your wishes guide you rather than fall under the gloomy shadow of humdrum practicalities. Of course there are limits to everything: cost, time, knowledge, technology. But why dwell on them? Why not pretend, just for a little while, that there are no limits to what we can do? It might just allow us to achieve what we thought we couldn't.

Take evolutionary design, for example. The earliest ventures into this field were made by engineers interested in optimization, and so most of the early systems were evolutionary optimizers. And very successful optimizers they were, too. Unfortunately, some ideas are so good that they can become conventional wisdom. Sometimes conventional wisdom can become tradition. And sometimes tradition can hinder freedom of thought. This is what happened for a while in evolutionary design. People became trapped by the idea that evolution optimizes designs, and so must only be used to optimize designs. If you're optimizing designs, then you need to parameterize an existing design, you need a concise and efficient fitness function and you need a very simplified design problem. Evolutionary design had been transformed from a nice idea into something that prevented us from tackling real-world design problems, or non-engineering design problems. You couldn't use this idea to do graphic design, or architectural design, or art, or even robot design. It wouldn't work. And so it doesn't get used by anyone.

Of course things have moved on since those days (although not everyone has caught up yet). Architects, artists and artificial life researchers have all bypassed the 'design optimization' legacy, instead creating their own approaches and their own traditions. The heady cocktail of evolutionary ideas we have available today are a blend of these techniques, resulting in software that can aid the creativity of designers with the same (or better) success than the optimizers optimize.

And yet we still struggle to remain free of the conventions and legacies from whence we came. Support from designers may be strong, but funding for these newer ideas is scarce. All too often the ideas must be toned down to camouflage them amongst the shades of gray of 'fundable projects'. The ambitions must be reigned in, for practical reasons. The goals must be made 'realistic' to meet time constraints. The successes must be limited.

It's all very depressing. So let's ignore practicalities. Let's let our imagination run free and see what emerges. Let's imagine a perfect system. A creative evolutionary design system that would fulfill every designer's dreams. A piece of software to make Bill Gates drool. A system that would become as essential to designers as a desk and chair. We haven't got the time or money to build it, but there's nothing to stop us from imagining it. And it might not take that much effort anyway. Maybe this system is only ten steps away.

2 TEN STEPS

STEP 1 *Find a domain in which it makes sense to use a computer for "creativity enhancement"*

It may be a perfect system, but we still need to know what it will actually do before we can make it. There is very little point in using creative evolutionary design system for standard function optimization - you need efficiency, not creativity, for that task. But there are many domains that are highly suitable. We know because we have had direct experience of using evolutionary systems for such applications. For example, Bentley has explored many related areas over the last few years. His first work investigated the creative potential of evolutionary design by enabling the generation of novel three-dimensional forms from scratch. Using a representation that permitted variable numbers of differently positioned and shaped ‘building blocks’ to be evolved, his GADES system showed how novel designs for tables, optical prisms, boat hulls and car body shapes can be evolved with great success. Subsequent work involved collaborations with architects to create evolutionary systems that could explore new 'inspirational forms', provide animations of the effects of imaginary future architecture or develop new hospital floorplans. Today, he is working with artists, biologists, computer scientists, ecologists and musicians to create evolutionary art, evolvable hardware, ecologies of novel plant forms and saleable compositions of music.

In contrast O'Reilly, a member of the Emergent Design Group at Massachusetts Institute of Technology (web.mit.edu/arch/edg/about.htm), has directed her attention towards conceptual Architectural form design. The Emergent Design group conducts research into architectural morphology and the emergent and adaptive properties of architectural form. The group's interactive software design tools exploit models of "natural computation" that can be viewed both as pragmatic, successful problem solving activities and, as generators of interesting and aesthetic physical or dynamic outcomes. The first project was called Generative Genetic Explorer (GGE). Implemented in AutoCAD, a genetic encoding modified by genetic operations defined structural spines of a "skinned" surface. The GermZ project combined Artifical Life concepts with evolutionary computation. A CAD version of a snowboard course was formed via the movement of evolving, breeding bacteria down a modelled slope with physical attributes and attractors and repellors. The Rule-Genetic Algomithm (RGA) followed. Written in Java and using the Java-3D toolbox, it provided a means of examining conflicting constraints between space usage and functional allocation. It also explored how modular elements could be joined to form emergent contiguous regions under local and central criteria. MoSS is an implementation of surface rendering in Alias|Wavefront Studio. A surface is defined by a3 dimensional L-system which yields conjoined plates when interpreted spatially. The interpretive process accomodates tropism by allowing environmental factors such as boundary and attractors to influence its definition of the surface plates. The most recent project is the Agency GP tool. It models the complex interactions of physical space and information technology within emergent organizations. It produces spatial systems and work environments. Implemented with Maya, Alias|Wavefront's latest surface modeller and animator, Agency extends a genetic programming paradigm with the innovative use of agents to determine design fitness. It also serves as a research testbed for interactive, design-based evolutionary computation.

Through research such as this, Bentley and O’Reilly have become familiar with the needs and requirements of ‘creative professionals’. It is clear that domains such as architecture, graphic design, art, musical composition, circuit design and indeed any problem that requires invention, imagination and exploration by a human are viable areas for assistance by a creative evolutionary design system (or CEDS).

STEP 2 *Find a good reason for using a creative system at all.*

"If it ain't broken, don't fix it," as the saying goes. There is little point in having a CEDS - even a perfect one - if it is not needed. If the designer(s) are able to keep up with the demands of their jobs, produce consistently original and good quality designs quickly, and don't mind being paid next to nothing, then a CEDS is probably superfluous. But such 'perfect designers' are usually difficult to find. There is usually too much work and not enough salary. Designers can become tired or bored, resulting in fluctuating levels of quality and originality. And sometimes the design problem is just too complicated for a designer to get to grips with alone. Design can be overwhelming, choices abound, decisions are interdependent, factors are non-linear. This is where our CEDS is needed. If the designer is overworked, the CEDS can assist by speeding up or even automating parts of the design process. If the designer is uninspired, the CEDS can spark the imagination. If the designer is unable to calculate the best compromise for a hundred different conflicting constraints and requirements, the CEDS can suggest solutions. And a creative evolutionary design system doesn't need a salary.

STEP 3 *Negotiate appropriately balanced control of the design process between the tool’s user and the tool itself.*

We can dream of a tool which looks over the screen of a designer while its sensors are hooked up to the designer's brain. Since the sensors can tap into the creative process, the tool knows exactly when to productively interrupt to make a good suggestion. The tool and designer work together seamlessly to arrive at a satisfactory design.

Control is not an issue in this dream. In today's reality it is. Face it, designers are control freaks. But deservedly so. Designers want to design, dammit! They don't really want a tool that can do their job independently of them. The ultimate tool helps a designer to maintain control yet exploit computation in order to produce better designs.

How can a CEDS negotiate this control issue? We must design the CEDS with a model of a car - that is a vehicle with automatic drive, steering wheel and brakes. The designer drives the car and has ultimate control. Yet, once started (by the designer), the CEDS advances according to the driver's direction with a series of nudges and direct interventions. The driver stops the car and determines its ultimate destination. We don't start a car and then fall asleep until it reaches our destination for us. So the perfect CEDS is *not* turn-key. It has to have user-friendly controls that allow interruption, intervention in the design generation process and resumption under changing goal criteria. Let us abbreviate this control negotiation as IIR - interrupt, intervene, then resume.

IIR contributes an important element to the relationship between designer and tool. It puts the designer in charge *when she chooses* while ensuring that the overall process (i.e. the process involving designer and tool) is a series of exchanges of control. The designer must be able to influence an ongoing design in order to make it appear the way she wants. She may wish to see what consequences her imposed changes have on the search trajectory through the design space. As the tool develops its outcome, she may wish to change the importance of different desired design properties or explicitly rule out some potential type of design. Ideally, to impose changes, she should to be able to use other tools or an enclosing tool. Then, the designer, *without starting over*, would like the tool to proceed again.

IIR as a facility can be broken into its three constituent steps when it comes to implementation. Interruption is a more or less straightforward mechanism to engineer. It may be controlled via a parameter available at the GUI. For example, in an evolutionary algorithm, the user can direct how many generations run before the search process stops and hands control of the best (or any selected) design over. Alternatively, a user can be given control via a mouse click that controls the tool's steps.

Intervention and resumption are interrelated. There are two means of intervention: indirect and indirect. With direct intervention, the designer actually alters a design and this design is returned to the tool. Resumption involves two tasks if direct intervention has taken place. First, the design changed by the user must be translated into the form of the internal representation and, second, the tool should proceed. With indirect intervention, the designer does not alter the design (or population of designs). Instead, she influences the computational subprocesses of the tool that, in turn, have an impact on the designs it produces. For example, in an evolutionary algorithm, the fitness function could be altered. Or, in a generative tool, an environmental property such as the strength of an attractor may be modified. If indirect intervention has occurred, resumption involves the tool proceeding with updated parameters.

Resumption is simply enacted with a crude input mechanism such as a key or mouse click which instantiates new values and continues the computation. Direct intervention is much thornier.

To delineate the challenges of direct intervention one first must consider the nature of a design's representations within the evolutionary design tool. Typically a design has both an internal representation (which may be encoded) and an external representation suitable for user presentation and evaluation. The task for direct intervention is to map user enacted changes to the external representation back to the internal representation because it is the internal representation that is used in the evolutionary process.

In some evolutionary algorithms (e.g. simple genetic algorithms, evolutionary strategies) the internal representation describes parameters that are used to elaborate a model (e.g. they are numerical coefficients of a geometric equation). The instantiation of the model with a specific set of parameters is the external representation. This makes mapping a designer-enacted change in the external represention simple if the designer changes something parameterized. But it is quite the opposite if she changes any non-parameterized aspect of the design, for it is impossible to make the reverse translation.

In other evolutionary algorithms (e.g. genetic programming) the internal representation is actually an executable structure. The executable structure is 'run through' an interpreter which results in the external design. Executable structures are extremely compelling because they provide more expressive flexibility than model-based parameterization. Direct intervention is hard to engineer in tools that use executable structures due to the interpretive process that the internal representation has passed through in being reformulated as the external design. If the user chooses to stop the tool and intervene with the presented design, there must be a means of backward-translating the updated design into a revised internal representation. Unfortunately the interpretive process is very difficult to reverse. Interpretation is not a one to one mapping so there could be many internal representations that generate an external representation. Which one should be chosen? Alternatively, a change the designer imposes may not even be expressable by the language of the representation and its interpreter.

O'Reilly's initial attempts to grapple with IIR came in the Agency GP tool (Testa, O'Reilly & Greenwold, 2000). In Agency once a population has been ranked by fitness, the tool becomes open to IIR. The entire population of interpreted designs is available for viewing by the user, who has several options. It is possible to indirectly intervene. The user can simply re-rank individuals (i.e., meddle with fitness values relatively) and allow evolution to continue. The user can also control what and how many pre-existing agents will be deployed to evaluate designs, or how heavily to weight each agent's findings. Agents also may have controls of their own which allow a user to direct their activities and thus indirectly readjust the discovery trajectory.

Alternatively, for the first time in our experience, direct intervention is possible. Agency implements a set of operations to modify an external design that can be reverse mapped to the design's internal representation (i.e., the representation manipulated by genetic operations). The user can select a candidate design and apply one or more of these operations derived from the

Agency language to an arbitrary number of the NURBS surfaces that comprise it. The transformations applied will be added to the list of operations in the internal representation of the individual. By providing the basic operations of three-dimensional modeling through our language we enable designers to make targeted modifications of designs before allowing evolution to continue. In evolutionary algorithm parlance (with apologies) we reverse map from phenotype to genotype by providing operations on the phenotype that we know in advance how to invert.

A rhetorical question is whether indirect intervention is better than direct intervention in a CEDS? Indirect intervention presents potential for frustration because it forces the tool user to try to influence the process that generates the design rather than allowing direct changes to the design. This 'nudging' quickly becomes tedious and is often non-intuitive. There are superficial ways to address this phenomenom but it exists precisely because the tools are based on evolutionary or generative algorithm concepts that require two levels of design representation.

For a tool to facilitate direct IIR, the altered external representation must be sent through the interpretation process backwards to obtain the internal representation that would specify it. However, this backwards process is not simple nor always possible.

For some applications, one may be willing to balance the difficulties of non-intuitivity and awkward usage of indirect IIR with the purpose and benefits of using evolutionary computation. For other applications, it is simply not acceptable. We think this is an important problem which will determine the ultimate benefit a CEDS can deliver.

STEP 4 *Find a specific niche in the design process for the CEDS, then make the tool accept its inputs and produce outputs that flow with the design process with its predecessor and successor modules.*

A perfect CEDS doesn't have to do 'everything'. It should be positioned as one little (but powerful) step in a bigger set of steps with the opportunity to be revisited often. Thus it needs to accept input and produce output in formats of the 'design pipeline'. One efficient and powerful way to do this is to deploy the CEDS within a useful, computerized design tool the designer already employs.

O'Reilly has made a rule of embedding the Emergent Design group's tools within existing CAD tools that architects are already comfortable with. This has not only leveraged software design (details of rendering and editing need not be implemented in the design tool) but it has allowed architects to import what they call 'site conditions' into the tool to configure it. It has also allowed the physical realization of the tool's (graphic) designs. The design outcomes of the design tools are directly outcomes of the CAD tool. Thus they can be saved in multiple formats and transferred to other software that allows laser cutting and surface reconstruction or even stereolithographic rendering (often called 3D printing).

STEP 5 *Make it generative and creative.*

Evolutionary tools are good at *generative* design and *creative, novel* design. Of course they make good optimizers too, and since we're thinking about a perfect system, we should allow our CEDS to optimize parts of designs if the designer needs such a utility. But, as we've seen, there are many types of design that need creativity and novelty. While we're quite experienced at optimization, creative computation is still a little tricky, so we'll focus on this aspect of the CEDS here.

Our perfect CEDS is under full control of the designer, and is so natural to use that the designer does not even realise she's using it. It needs to monitor the creative output of the designer and fill in any deficiencies: providing ideas on the slow days, helping to speed up the output of the designer on the designer's good days. To achieve these things it must be generative: able to build upon existing ideas, or even create brand new ideas from scratch. It should be able to suggest bizarre ideas to help inspire the designer, or to make them realize that different alternatives exist.

Thankfully, this part of the CEDS has already been demonstrated in a number of existing systems. These tools attack much larger design spaces than traditional optimisation approaches. Their invisible internal representations have had many constraints removed, enabling a huge range of design solutions to be defined for every new problem. These systems do *not* take an existing solution, parameterise a small part of it, and then optimize those parameters. Instead they often begin with nothing but a collection of components (which might be 3D shapes, GP functions or electronic components), and it is up to the CEDS to pick and organize those components to generate a solution. By using such component-based representations, these systems explore the searchspace for new ways of assembling solutions, rather than optimize existing solutions (Bentley & Corne, 2001). Novel art, architecture, design and even music are commonly evolved using these representations, the whole being built from generations of slow additions, deletions and shaping of separate components.

Bentley's work in this area has investigated a variety of different component-based representations. His GADES system used components made from variable numbers of separate 3D clipped stretchable cuboids, permitting almost an infinite number of different shapes to be defined. Later work used a combination of GP with fuzzy logic to perform fraud-detection by generating novel rule sets. In this case the components were fuzzy-logic operations such as AND, OR and IS_HIGH. Current work includes the generation of music by a genetic algorithm, using components made from musical notes and drums.

Increasingly, Bentley's work is focussing on *hierarchical* component-based representations – especially those that arise spontaneously during genotype to phenotype mapping. By making use of many of the tricks employed during biological developmental processes, our designs can be 'grown' from their genotypes. When the genomes comprise sets of growth instructions defining how components should be assembled, it is possible for evolution to develop its own hierarchies, duplications and subroutines in the corresponding phenotypes. And when most real-world designs contain exactly such features, the use of computational embryology may play an important role in future CEDSs. Instead of needing to hand-design appropriate internal genetic representations which define the necessary hierarchical structures of components, the system could evolve its own hierarchies.

So we now potentially require three separate representations: a genetic representation of 'developmental rules', a component-based representation which is used to build the solution (following those rules) and perhaps even a separate phenotype representation, to enable fast evaluation. Such creative evolutionary design systems have already been built (Bentley and Corne, 2001) and are showing the potential for improved creative design.

The only downside of such approaches are the implications they have for IIR. When the final designs are so far removed from their genotypes, it becomes considerably more difficult to derive genotypes from phenotypes supplied or changed by the designer. Potentially, an evolutionary process could be used to discover the new genotype (using the changed phenotype as its target). Realistically, however, the use of improved (developmental) component-based representations seems likely to restrict the designer to indirection intervention during evolution.

STEP 6 *Make it understandable.*

A perfect CEDS provides its users with a generative, creative process metaphor that is not over-encumbered by evolutionary details and yet is accurate enough to allow them to have a working understanding of it. If users are required to understand genetic representations, the meaning and impact of evolvability, and the gory details of crossover, mutation and genotype to phenotype mapping, then their jobs become less about designing and more about evolutionary computation. Our CEDS must keep designers doing what they are best at: designing. We must not turn them into computer scientists.

Thankfully, evolution can work very well as a 'black box' technique. Genetic representations, fitness functions and all the underlying machinery does not need to be visible to enable a CEDS to work. Indeed, because of the steep learning curve and mystic artform that still comprises genetic encoding in EAs, a small amount of knowledge of these issues can be more harmful than none at all. Instead, users need only know what they already know: designs are in families, child designs resemble parent designs with some variation, some designs work better or look better than others. This is the way our own design processes generate designs (and most other products of the human mind). It is an eminently understandable and simple concept for a computer system - especially compared to the extraordinary complexity of most computer aided design and visualisation packages.

STEP 7 *Have an easy and effective way of evaluating the quality of solutions and guiding the path of evolution.*

In some respects, optimization is much easier than creative evolution. Fitness functions is one of those cases where this is true. Although calculating, programming or interfacing to evaluation software can be very difficult, at least there is some kind of clearly-defined method of allocating fitnesses. For creative designs, the whole notion of fitness can be very difficult to pin down. In architecture, for example, a good design may be far more expensive, difficult to build and impractical than a bad design, purely because the good one is being judged on aesthetics, or planning laws, or by clients in a committee. Often the evaluation criteria consist of a million different objectives and constraints, taught to designers through years of training and experience and never written down in any explicit form.

In the Agency GP tool we decided to let the architects model the conflicting, non-linear, interdependent design criteria via distributed 'software agents' inhabiting the candidate designs (Testa, O'Reilly & Greenwold, 2000).

Virtually any criterion for evaluation can be coded and dropped in as an agent to our framework. We can specify that workspaces require a certain quotient of natural light or that circulation spaces desire width enough to allow for conversation. Using agent-based evaluation, we will able to model management structures and determine their influence on potential designs. An agent may represent the pattern of a group, its needs for privacy, meeting space and collaborative surfaces, or it may undertake the concern of management structure or productivity. Agents individually and collaboratively rate the design, and their feedback is incorporated into a measure of overall fitness.

Agents are not enough because they force explicitization of the tool user of aesthetic preference. Such self-awareness may not be possible or may be inaccurate. Our perfect CEDS needs to learn, capture and allow the input of these objectives to enable the evaluation of new creative designs.

Knowledge elicitation is the traditional way to embed external information from experts into a computer system. Fuzzy logic rules enable English-like sentences to describe precise objectives and constraints, and have been used with some success to add to fitness functions (Soddu, 1995; Parmee, 1999).

One particularly good source of information about good designs is - good designs. Existing designs were created by designers to meet all of the same complicated objectives that the CEDS has to meet. So knowledge about what works is embedded into all good designs. There are a number of ways in which this can extracted: seed the initial population with some of these designs, so that the CEDS can mix and match the good ideas (Rosenman). Or use an evolutionary algorithm to learn the styles employed in a group of designs, providing the CEDS with a representation able to define designs in that style (Gero and Kazakov, 1996). Or use some of the existing designs as targets, and rate new evolving designs on how closely they match aspects of the targets. And this approach is not limited to existing designs - if the CEDS finds a particularly good design at any time, it can be stored in "digital amber" as Steven Rooke calls it, enabling it to be used in the future - a very common technique in evolutionary art systems (Bentley & Corne, 2001).

And finally, unlike most evolutionary paradigms, a CEDS will often need user-interaction to guide evolution. We have already seen how IIR is essential for the proper integration of the tool with the designer's work. But interaction in the form of selecting good designs for reproduction, and removing bad ones, is also essential to allow all of the designer's unwritten knowledge to be expressed. To achieve this in an easy-to-use way requires a good GUI to permit visualization of the current designs, and allow better designs (or better parts of designs) to be rated by the designer. Many such schemes exist in evolutionary art systems, often requiring artists to click and judge grids of images with the mouse. Our perfect CEDS must use this type of interface where necessary, but also use it sparingly, for boredom and fatigue quickly dull the judgement of anyone who has to rate too many solutions.

STEP 8 *Find people who are actually prepared to use the system.*

Even those designers who actively participate in the development of a CEDS may be reluctant to use the system once it is complete. From experience, designers (and all those who feel that they express an important part of themselves through their work) are very slow to take up new technologies, even if those techniques are designed to assist rather than replace them. It can be one of the most frustrating parts of research projects: developing an amazing tool that is never used by the people it was developed for. Finding people to use our perfect CEDS may be more about education than search. Once designers have seen the benefits and tried it for themselves, their technophobia should be reduced. Anything this fun to use cannot be feared.

STEP 9 *Get lots of money to pay R&D costs.*

It can still be difficult to obtain funding for this kind of research. Because CEDSs are so new, few have been demonstrated and even fewer are currently being used by designers. As this field matures and the ideas become more meanstream, funding will inevitably follow.

STEP 10 *Start a company and make a billion.*

A good measure of success of any computer system is whether it is successful enough to make a profit in the market place. If our perfect CEDS is ever built, then maybe it could be that successful. Only time will tell.

3 SUMMARY

In this position paper we have taken a step-wise look at developing a perfect creative evolutionary design tool. We've tried to include a survey of shortcomings and solutions to issues that arise in achieving such perfection. Of course such a brief article cannot cover every aspect of creative evolutionary design, so we welcome ideas from others concerning issues we've neglected. We'd also love to hear even better ways to resolve them.

Acknowledgements

Una-May O'Reilly thanks the members of the Emergent Design Group: Devyn Weiser, Peter Testa, Simon Greenwold and Martin Hemberg. Peter Bentley thanks the continuing support of the Department of Computer Science, University College and especially the members of the nUCLEAR group.

Bibliography

Further details of all the projects by Bentley and O'Reilly mentioned in this article can be found on the web pages:

http://www.cs.ucl.ac.uk/staff/P.Bentley/PetersPapers.html

http://www.ai.mit.edu/people/unamay/papers

Bentley, P. J. and Corne, D. W. (Contributing Eds.) (2001) *Creative Evolutionary Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA (to appear).

Gero, J. S. & Kazakov, V. (1996) An exploration-based evolutionary model of generative design process. Microcomputers In Civil Engineering 11, 209-216.

Rosenman, M. & Gero, J. S. (1999) Evolving Designs by Generating Useful Complex Gene Structures. Ch 15 in P. Bentley (Ed.) *Evolutionary Design by Computers*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 345-364.

Testa, P. and O'Reilly, U.M. and Greenwold, S. (2000). Agent-Based Genetic Programming for Spatial Exploration. In Proceedings of ACSA.

Soddu, C. (1995) Recreating the city's identity with a morphogenetic urban design. *17th International Conference on Making Cities Livable*, Freiburb-im-Breisgau, Germany, Sept. 5-9 1995.

Parmee (1999) Exploring the Design Potential of Evolutionary Search, Exploration and Optimization. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., San Francisco, CA, 119-143.