Exploring Component-based Representations - The Secret of Creativity by Evolution?

Peter J. Bentley
Department of Computer Science, University College London
e-mail: P.Bentley@cs.ucl.ac.uk

Abstract. This paper investigates one of the newest and most exciting methods in computer science to date: employing computers as creative problem solvers by using evolution to explore for new solutions. The paper introduces and discusses the new understanding that explorative evolution relies upon a representation based on components rather than a parameterisation of a known solution. Evolution explores how the components can be arranged, how many are needed, and the type or function of each. The extra freedom provided by this simple idea is remarkable. By using evolutionary computation for exploration instead of optimisation, this technique enables us to expand the capabilities of computers. The paper describes how the approach has already shown impressive results in the creation of novel designs and architecture, fraud detection, composition of music, and creation of art. A framework for explorative evolution is provided, with discussion of the significance and difficulties posed by each element. The paper ends with an example of creative problem solving for a simple application - showing how evolution can shape pieces of paper to make them fall slowly through the air, by spiraling down like sycamore seeds.

1. Introduction

Similar to the field of neural computation, evolutionary computation comprises a set of techniques inspired by natural processes. Instead of being modelled on the workings of brains, evolutionary algorithms are based on natural evolution. Problems are solved by using populations of solutions that ‘reproduce’, and ‘die’ according to how well they satisfy the problem, resulting in the emergence of good solutions after a number of generations.

Despite the first evolution-based programs dating back to the 1960's, this subset of computer science has only in the last ten years become recognised as a field of research in its own right. Today, hundreds of researchers world-wide, numerous departments and over twenty international conferences a year devote their efforts to understanding and applying evolution to problems.

The steady increase of interest in evolution-based approaches seems to be due to the fact that evolution provides a quick and easy way to solve difficult problems. It is now commonplace to evolve solutions for scheduling, machine learning, anomaly detection, ordering problems, data mining, control, game playing, reliable and fault tolerant systems, design optimisation and many other problems [1]. Evolutionary biologists now use evolutionary algorithms to increase understanding of natural evolution [2].

Evolution allows us to overcome many of the problems associated with searching for difficult and complex real-world solutions. But traditional implementations of evolutionary search suffer from the same fundamental drawbacks as all conventional search algorithms. They rely on a good parameterisation to permit them to find a good solution. If we are optimising a propeller blade, but the parameterisation does not permit the width of the blade to vary, then the computer will never be able to find solutions with different widths. If there is no parameter for something, then the computer cannot modify it. Evolution, like all search algorithms, is limited and constrained by the representation it can modify.

However, recent work has removed many of these limits and constraints. We can now use evolution to explore further than ever before. Evolution can search for good search spaces, even as it searches within a space. It can alter the dimensionality of a space, the parameterisation, the representation of solutions, and at the same time, find good solutions in this ever-changing hyperspace. Evolution can be used to modify so much of the problem that the very concept of search space begins to become unhelpful. The advantages of such freedom of search are plain: the evolved solutions now resemble inventions rather than improvements [3]. And because so many of the constraints of representation are thrown away, our computers begin to seem creative in the way they arrive at these surprising results.

2. Explorative Evolution

Most of the advances in explorative evolution have grown up on my home turf, evolutionary design [1]. And it is easy to see why this has happened. Design problems such as architecture, engineering design, and aesthetic design are horribly complex, with huge numbers of (often conflicting) objectives, many constraints and often thousands of parameters [4]. But the most difficult aspects of these design problems are *people*. Designs are usually used by people - we live in architecture, we use and interact with the things around us. We like and dislike things almost at random, and as fashions change, so our preferences and requirements change. This means that a design specification will usually be a moving target [5]. It will have many unknowns, and the few things that are true one week will not necessarily be true the next week.

Designers take such things for granted. They know that their designs will be revised and modified many times until clients are satisfied (or until time or costs prevent further changes). And architecture is perhaps the most extreme type of design in this respect. There are so many different rules, opinions, preferences and materials that for every new building, that there will be an infinite number of possible design solutions. Exploring these solutions forms part of the difficult job of being an architect. Consequently, it is no coincidence that the first forays into explorative and generative evolutionary design were made by architects. Some of the earliest work was performed by Prof. John Frazer, who spent many years developing evolutionary architecture systems with his students [6]. He showed how evolution could generate many surprising and inspirational architectural forms, and how novel and useful structures could be evolved. His methods often involved the use of components such as cellular automata, which were evolved and sometimes wrapped in surfaces to generate smooth exteriors. In Australia, the work of Prof. John Gero and colleagues such as Dr. Michael Rosenman (both architects) also investigated the use of evolution to generate new architectural forms. This work concentrates on the generation of new floor-plans for buildings, showing over many years of research how explorative evolution can create novel floor-plans that satisfy many fuzzy constraints and objectives [7]. They even show how evolution can learn to create buildings in the style of well-known architects [8].

Designers and architects still remain at the forefront of this area of research.

Today increasing numbers are creating evolutionary architecture systems capable of generating novel forms, structures, buildings and even towns. For example, Paul Coates of the University of East London has shown how evolution can generate coherent plans for housing estates and buildings, as well as innovative building exteriors [9]. Prof. Celestino Soddu of Italy uses evolution to generate everything from novel table-lamps to castles to three-dimensional Picasso sculptures [10]. The work of Dr. Ian Parmee at the University of Plymouth has revealed a variety of methods for handling the complex and fluid nature of engineering design problems over the years [4].

Artists are also keen researchers into explorative evolutionary systems. The use of evolution to generate form, judged entirely on aesthetics, was first shown by Prof. Richard Dawkins in the mid 1980's [11]. His work inspired the well-known work of William Latham and Stephen Todd, who had considerable success in evolving artistic images and animations [12]. The work of Karl Sims was also inspired by Dawkins, and also showed the astonishing complex and aesthetically pleasing images that evolution could generate [13]. Evolutionary art systems have now become very popular, with many programs available and some now being incorporated into art and CAD packages [14].

All of these examples of evolutionary systems used evolution as an explorer, not as an optimiser. Normally guided by a human, the software is used to investigate many possible solutions, to provide inspiration and to give a feel for the range of useful solutions. Whilst producing impressive results, such software always received some criticism that the presence of a human to guide the direction of search by evolution was the key to the success of these systems. My own work in this area, and the work of others such as Adrian Thompson and John Koza, provided some of the first convincing demonstrations that evolution was capable of innovation without human guidance. I showed how evolution would find surprising and creative solutions to design problems, even when only software guidance was provided. By telling the computer the desired function in the form of a set of evaluation routines, but not anything about the design itself, I removed the human from the loop and showed conclusively the power of explorative evolution [15],[16]. Adrian also demonstrated the power of evolution, this time to generate novel electronic circuit designs which were tested using field programmable gate arrays [17]. In a similar vein, John Koza has been demonstrating the use of genetic programming to find novel computer programs for some years, i.e. to use computers to program themselves [18]. Work such as this began the recent change in thinking about evolution. No longer were we content to regard an evolutionary algorithm as 'another optimiser'. Our evolutionary programs were now independently solving problems for us, and finding creative solutions that surprised us.

Since then, research in this area has expanded rapidly. Adrian Thompson spends his time trying to develop ways to analyse how his evolved circuits work [19]. Julian Miller has expanded upon Thompson's work, showing how evolution can create circuits that seem to work on entirely new principles [20]. John Koza has recently announced the completion of a 1000-processor Beowulf computer which will be used for the sole purpose of 'using genetic programming as an automated "invention machine" for creating new and useful patentable inventions'. His group has recently demonstrated the use of evolution to generate many different types of analogue circuit, many of which mirror or outperform our best human-created circuit designs [21]. Jordan Pollack has shown how evolution can generate highly novel structures such as bridges and cranes [22] and has now begun to use these methods for the evolution of robot bodies. Karl Sims used evolution to create the bodies and brains of ‘virtual creatures’ capable of swimming, walking, jumping and competing in virtual environments [23]. Husbands et al [24] used evolution to generate novel propeller-like forms. Many research departments around the world use evolution to generate new and highly complex neural networks for various applications. My own work in this area continues, as I use evolution to compose music that cannot be distinguished from human compositions [25]. (For a more comprehensive review of work in this area, see [26]).

3. Exploring the Explorer

The research described here aims to investigate explorative evolutionary computation. To date there has been no significant research aimed at understanding the difference between exploration and optimisation. There has been little increase in our understanding of how evolution can innovate since we began demonstrating this ability. This project aims to discover answers to these fundamental questions and to explore, characterise and explain how we can use these answers to further increase the capabilities of explorative evolutionary computation.

The first question to be tackled is the most fundamental: what is evolution doing when it explores, that it is not doing when it optimises?

Traditional views of an evolutionary algorithm regard this search technique as an optimiser. A better term is actually ‘satisficer’ [27]. Evolution never tries to find globally optimal solutions. It merely propagates improvements through the population. In doing so, evolution walks a mysterious and winding path through the search space. Sometimes this path may reach a dead-end (premature convergence). Sometimes the path may go around in circles. And occasionally the path may lead to a global optimal - but there is no guarantee of this. The wanderings of evolution are like those of an explorer.

But exactly what does evolution explore? This is determined by the representations it uses. With a fixed-length parameterisation, explorations do resemble optimisation. For example, if there are only three parameters defining the solution and a single fitness function, evolution will behave in much the same way as an optimiser - it will find the best values for those three parameters such that the solutions are as fit as possible. But with a different kind of representation, the behaviour of evolution changes. When the parameters do not define the solution directly, when they define a set of components from which the solution is constructed, the idea of optimisation becomes inappropriate. Now evolution *explores* new ways of constructing the solution by changing the relationships between components. It can vary the dimensionality of the space by adding or removing elements. It can explore alternatives instead of optimising a single option. When we examine the representations, objectives and goals of optimisation and exploration, the difference between these approaches becomes clearer:

Optimisation: A knowledge-rich encoding of the problem is used, i.e. a solution is parameterised using the fewest possible parameters (and thus both minimising the search space, but also minimising potential for exploration). Evolution is used only to find the best parameter values within that parameterisation. Objective functions are normally predefined and fixed. The goal is to find a global or near global optimum with minimal computation.

Exploration: A knowledge-lean representation is used, and instead of a parameterisation of a solution, a set of low-level components is defined. Solutions are then constructed using the components, allowing exploration (sometimes at the expense of size of search space and ability to locate optima). Objective functions may vary at any time. The goal is to identify new and interesting solutions - normally more than one is desirable - and these solutions must be good. However, finding global optima may be undesirable, impractical or even impossible.

This difference between optimisation and exploration is rarely considered, let alone defined. And yet when the literature in this area is examined, it becomes evident that every system that performs exploration uses this component-based representation. Researchers who evolve architecture use evolution to manipulate components such as walls or bricks. Researchers who evolve electronic circuits use evolution to modify components such as logic gates, transistors and capacitors. Artists use evolution to create art out of primitive shapes: swirls, spheres, tori, or out of mathematical functions such as cosine, arctangent and factorial. Likewise, computer scientists using GP evolve programs from components contained within the function and terminal sets. My own work used cuboid blocks to construct designs, and currently uses fuzzy logic functions to construct fuzzy rules and musical notes to construct compositions. Every explorative evolutionary system relies on some kind of component-based representation. Figure 1 illustrates this idea, showing how components allow increased freedom for evolution.

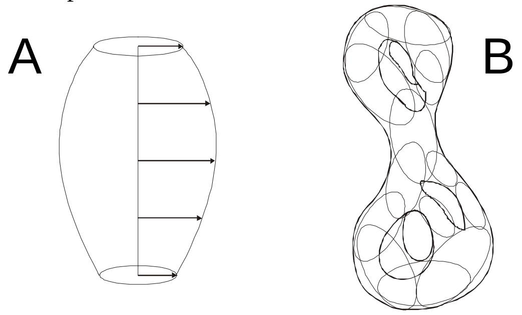

Figure 1 Optimisation of fixed parameterisation versus component-based exploration. Shape A uses minimal parameters and is knowledge-rich (the height of the shape and the swept 2D outline is assumed by this representation). Shape B is constructed by wrapping an envelope around a collection of 3D ellipses. By varying the relative positions and sizes of the ellipses, vastly more innovative and creative forms can be generated.

Evolutionary algorithms, and in particular genetic algorithms and genetic programming, naturally lend themselves to this kind of exploration. It is standard practice to use evolution to manipulate coded versions of solutions (genotypes) and to map these onto actual solutions (phenotypes). This use of genotypes and phenotypes means that the distinction between, and use of, components of solutions and complete solutions is natural to this field.

4. A Framework for Explorative Evolution

From this novel intuitive understanding, a framework for explorative evolutionary systems can be constructed, containing the following five components:

1. An evolutionary algorithm.
2. A genetic representation.

3. An embryogeny using components.
4. A phenotype representation.
5. Fitness function(s).

To summarise, an explorative evolutionary system requires some kind of evolutionary algorithm to generate new solutions. The algorithm modifies genotypes defined by the genetic representation, which must be designed to minimise disruption caused by the genetic operators. An embryogeny (or mapping process) must decode the genotype, and using some kind of components, must construct the phenotype. The phenotype representation must be designed such that it permits quick and efficient evaluation by the fitness function(s). It is likely that the evolutionary algorithm, the genetic representation and to some extent the embryogeny, will be generic and suitable for reuse for most problems without modification. The phenotype representation and fitness functions must be specific to the current application of the system. The following sections explore the elements of the framework for explorative evolutionary systems in more detail.

Evolutionary Algorithm: The evolutionary algorithm forms the core of any evolutionary system. There are four main EAs in use today: the genetic algorithm, genetic programming, evolutionary strategies and evolutionary programming. Only the GA and GP are commonly used for explorative purposes. The reason for this can be found in the way these algorithms work. The genetic algorithm maintains genotypes and phenotypes, with a mapping between the two. As described earlier, this distinction has helped to encourage some GA researchers to use component-based genotype representations that map onto the phenotype representations, thus allowing explorative evolution to begin. In the same way, genetic programming also makes use of genotypes (this time with tree-structures) that are mapped onto phenotypes such as programs, images or circuits. GP has the advantage that its genetic representation *requires* the use of smaller components (in the function and terminal sets), so all applications of GP demonstrate the explorative power of evolution. This explains why the first notion of “invention machine” came from John Koza, the inventor of GP - his algorithm ensures that explorative evolution will always take place. In contrast, algorithms such as evolutionary strategies and evolutionary programming make no distinction between genotype and phenotype. By directly modifying the solution and with no provision for mapping to new representations, these approaches make the use of components to construct solutions more difficult to implement - but not impossible.

Within any evolutionary algorithm there are other issues that must be tackled. Handling multiple objectives, multimodality, noise, premature convergence, fuzzy or changing fitness functions must all be considered. Solutions to all of these problems, using ideas such as Pareto optimality, region identification, speciation, variable or directed mutation rates and steady-state GAs are now emerging in evolutionary computation [26], [4], [28]. These issues, although important, are not the most significant consideration for explorative evolution. Indeed, even the choice of evolutionary algorithm (or indeed any other search algorithm) is secondary to the representations, for it is the representations that permit evolution to explore.

Genotype Representation: The genotype representation defines the search space of the algorithm. A poor representation may enumerate the space such that very dissimilar solutions are close to each other, making search for better solutions harder. For explorative evolutionary computation, where genes will represent (directly or indirectly) a variable number of components, the search space is typically of variable dimensionality, thus making its design even harder [7].

There are also other problems. Because of the use of components to represent solutions, the likelihood of epistasis dramatically increases. Not all component-based representations will have this effect (e.g. a voxel representation allows both exploration and zero epistasis). However, most components are inherently linked to their companions for the solution to work as a whole. A circuit relies on the links between its components, a melody relies on links between notes, a house relies on links between walls, a program relies on links between commands. These linkages all mean that corresponding genes become epistatically linked, resulting in potentially serious problems for evolution. With polygeny so prevalent in these problems, great care must be taken in the design of the genotype representation and corresponding genetic operators to minimise the disruption of inheritance.

Practitioners of GP have long been aware of these problems, with many solutions now in existence. Modifications can be made to the genetic representation to increase functionality and decrease disruption, e.g. ADFs, ADIs, ADLs, etc. [21]. Genetic operators that enforce typing help ensure that genetic trees are not shuffled too drastically during the production of offspring [29].

GAs do not require the use of tree-structured genotypes, so genetic representation-based problems are often less prevalent. GAs can be used to evolve variable-length genotypes and structured genotypes, typically with operators designed to perform crossover only at points of similarity between two parent genotypes, for example [30]. Advanced GAs designed to minimise damage caused by disruption of epistatic links between genes have also been demonstrated [31]. In addition, GAs do not suffer from the classic GP problem of bloat, where genotypes tend to increase in size, with redundant genetic material becoming ever greater in solutions. It is clear that the creation of suitable genetic representations and corresponding operators is a considerable problem in its own right. Furthermore, recent research seems to indicate that significant benefits may be gained from using less complex genetic representations and operators, instead making use of embryogenies of greater complexity [32].

Embryogeny: An embryogeny is a special kind of mapping process from genotype from phenotype. Within the process, the genotype is now regarded as a set of ‘growing instructions’ – a recipe which defines how the phenotype will develop. Polygeny is common, phenotypic traits being produced by multiple genes acting in combination. My own research in this area has revealed some of the potential of these advanced mapping processes. Advantages include reduction of the search space, better enumeration of search space, the evolution of more complex solutions, and adaptability [26].

Embryogenies are widely used for explorative evolutionary systems, for they provide the mechanism for constructing whole solutions from components. Three types of embryogeny are used today, the first and most common being *external*, where a programmer writes the software that performs the mapping, and the process cannot be evolved [26]. More recently, *explicit* embryogenies have become popular, with every step of the growth process explicitly held as part of the genotype, and evolved [26]. Examples include Cellular Encoding (used by Koza and team for the evolution of analogue circuits [21], Lindenmayer Systems (used by Coates for the evolution of architectural forms [9] and shape grammars (used by

Gero for the evolution of floor plans [8]. Despite the considerable success of these embryogenies, they often require complex additions to genetic representations and operators to allow evolution to work. The third type of growth process, the *implicit* embryogeny, has shown the most exciting results and greatest potential in recent work. Instead of evolving the mapping as a set of explicit steps in the genotype, an implicit embryogeny uses a set of rules, typically encoded as binary strings in a GA genotype. For each solution, a 'seed' component is created, and then the rules are iteratively applied. Over many iterations, with rules activating and suppressing each other, the growth, position, and type of new components are built up, finally resulting in the development of a complete solution. This emergent growth process shows remarkable properties of scalability, with the genotype describing solutions of increasing complexity without any increase in the number of rules needed - the rule-directed growth process is simply allowed to run for more time. This is in contrast to all other approaches, which require significantly larger genotypes to define the increased growth of more complex solutions [32].

The process of mapping from genotypes to phenotypes is clearly of importance to the investigation of explorative evolution. Issues of scalibility, evolvability, and biases induced in search have yet to be considered by researchers in any great depth. Increased understanding in this area would benefit both computer science and developmental biology.

Phenotype Representations: Once constructed by the embryogeny, the resulting solution is defined by the phenotype representation. Typically this representation is application-specific - if we are evolving circuits, the representation might define networks of connected components, if we are evolving buildings, the representation might define exterior shapes and/or interior walls, floors and stairs. An important criterion is evaluation - typically the phenotype representation will be designed to allow direct evaluation by fitness functions, without intermediate transformations or calculations. A poor choice will detrimentally affect processing times and solution accuracies.

The distinction between genotype, embryogeny and phenotype representations is often blurred in this field. Some GP practitioners regard all three to be the same. Others, such as Jakobi's work on evolving neural networks [33] or Taura's work on evolutionary configuration design [34], use different and distinct representations for each stage. It is still unclear whether the component growing process of the embryogeny should use the same representation as the phenotype - should the phenotype be represented by blocks or should the blocks be merged into a single, whole description of the solution? For example, should an evolved musical composition be represented by a series of notes or by a single, complex waveform in the phenotype? The answer is likely to depend on the fitness functions.

Fitness Functions: The fitness functions must provide an evaluation score for every solution. For explorative evolutionary computation, this is often a little harder. Typically the use of components rather than a parameterised solution means that early results can be chaotic to say the least. A design of a car may resemble a shoe; a melody may sound like a burglar alarm. Before evolution has had time to improve these initially random solutions, they can be nothing at all like the desired result. And yet the fitness functions must always be able to provide a fitness score that makes sense. The task is made even harder when unknowns or approximations must be incorporated into the evaluation, or when constraints and objectives are varied to aid exploration. Potential solutions include the use of custom-designed modular functions [16] and fuzzy logic [4] to cope with such problems.

Many explorative systems use human input to help guide evolution. Artists can completely take over the role of a fitness function [12], [13], and more recent work has investigated the use of these techniques for evolving photo-realistic images of faces for the identification of criminals [35]. These applications raise numerous human-computer-interfacing issues, i.e., will an explorative system detrimentally affect the style of an artist or the memory of a crime victim? These software tools have been shown to aid imagination and creativity, but how best to let the user inform evolution of his/her preferences and how best for the computer to report the structure and contents of the space being explored? Clearly, further research is required to address these issues.

5. Example Creative Application

To demonstrate the potential of explorative evolution, this section of the paper describes a simple application **-** the generation of shapes that, when constructed out of paper, fall as slowly as possible to the ground. A simple explorative evolutionary system was developed, following the framework provided above. A 'simple GA' was used as the evolutionary algorithm [36]. The genotype representation comprised a flat chromosome of 16 binary coded genes. A very basic mapping process or embryogeny was used to derive 8 (x, y) parameter values from each chromosome, join each vector to its successor by an edge, join the last vector to the fist with an edge, and fill the resulting shape. This process was performed by executing PostScript instructions output by the system, printing the shapes, and using a scalpel to cut out the shapes. The resulting paper phenotypes ('represented' by reality) were then tested by releasing them from a height of 150mm three times in succession. Each phenotype was allocated their fitness score by calculating: 1/(time1+time2+time3), ensuring that evolution would attempt to generate phenotypes with increased 'falling times'.

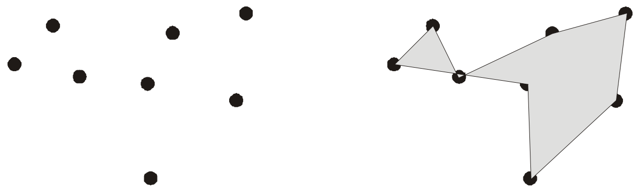

Figure 2 The *paper fall* application uses eight vertices as its components (left). These are extracted from the genotype and transformed into real paper shapes by the use of a 'join the dots and fill the shape' embryogeny (and someone to print and cut out the shape).

This application illustrates the use of an extremely basic component-based embryogeny representation. As figure 2 illustrates, the components are simply eight vertices with (x, y) positions. Despite these components having no size and no type, the unconstrained freedom of position of each vertex relative to all other vertices means that this component-based representation allows the definition of a vast number of different shapes. Clearly this is a knowledge-lean representation (no information about which shapes are best is provided). It should also be clear that the representation allows evolution to explore the solution space in an unconstrained manner, and that there will be many millions of good and bad solutions for this problem.

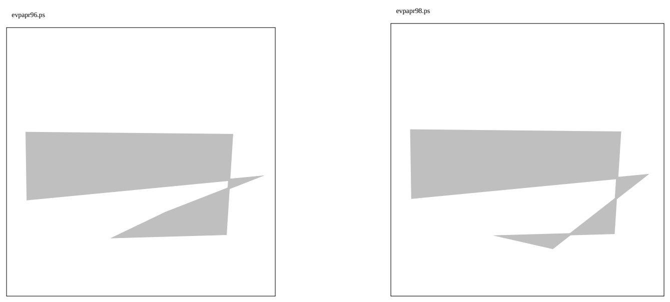

Figure 3 Two paper shapes evolved to fall slowly through the air. Both are members of the final generation, and both use a smaller 'arm' or flap to cause the shape to rotate as it falls, like a sycamore seed. (Not shown at scale used for testing.)

Predictably, the use of real-life testing is time-consuming and laborious (especially if you make the mistake of performing the experiment yourself instead of enlisting the services of a student). Because of this, population sizes of 10 individuals were used in an evolutionary run of 10 generations. Despite these excessively low values, evolution was able to make significant improvements on the time taken for the shapes to reach the ground. Times taken for initially random shapes to fall 150mm varied from 0.7 seconds to 1.8 seconds. By the tenth generation, all shapes took, on average, more than 2 seconds to fall the same distance. Figure 3 shows two of the solutions in the final population. Convergence has begun to occur, with most shapes using the same technique of having a smaller flap which causes the shapes to rotate as they fall. The main benefit of this solution appears to be the way rotation stabilises the motion of the shape, preventing it from slipping sideways in the air and plummeting to the ground at tremendous speed. Even with the very limited resources evolution was given, a 'creative' solution to the problem was found.

6. Conclusions

Creativity is a very hard concept to define, but this paper does not suggest that the creativity of people depends on the exploration of component-based representations. However, evolution is very different from the human brain. It is apparent that all of the evolutionary systems that claim to produce 'creative' or innovative results do rely on the use of component-based representations with evolution. It is also clear that such unconstrained and knowledge-lean representations do provide far greater freedom for evolution to innovate, compared to knowledge-rich representations based on a parameterisation of a solution.

With this insight into enabling creativity by evolution, we can create a framework for explorative evolutionary systems, comprising: evolutionary algorithm, genotype, embryogeny and phenotype representation and fitness functions. Although looking familiar, the framework employs a component-based embryogeny to map genotypes into phenotypes, which raises more unusual issues for the genotype and phenotype representations and the fitness functions.

This framework was illustrated by the *paper fall* application, which used binary strings for the genotype representation, a component-based embryogeny using eight vertices to form shapes, and real phenotypes constructed from paper. Even using such simple techniques, evolution was able to generate some creative solutions in the form of 'sycamore seed' shapes, demonstrating the power of explorative evolution. Although it may not be the only reason for the success of evolution for these problems, it seems that exploring components is one of the secrets of creativity by evolution.

References

1. Bentley, P. J. (Contributing Editor), 1999a. *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., San Francisco, CA.
2. Dawkins, R., 1996. *Climbing Mount Improbable* Penguin Books, Ltd..
3. Bentley, P. J. and Corne, D. (Eds), 1999. Proceedings of the *AISB'99 Symposium on Creative Evolutionary Systems (CES)*. Published by AISB, Sussex, UK. ISBN 1 902956 03 6.
4. Parmee, I., 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.
5. French, M., 1999. The Interplay of Evolution and Insight in Design. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc.
6. Frazer, J., 1995. *An Evolutionary Architecture*. Architectural Association, London.
7. Gero, J. S. & Kazakov, V., 1996. An exploration-based evolutionary model of generative design process. *Microcomputers In Civil Engineering* 11, 209-216.
8. Schnier, T. and Gero, J. S., 1996. Learning genetic representations as alternative to hand-coded shape grammars, in J. S. Gero and F. Sudweeks (eds), Artificial Intelligence in Design'96, Kluwer, Dordrecht, pp.39-57
9. Coates, P., (1997) Using Genetic Programming and L-Systems to explore 3D design worlds. *CAADFutures'97*, R. Junge (ed), Kluwer Academic Publishers, Munich.
10. 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.
11. Dawkins, R., 1986. *The Blind Watchmaker*. Longman Scientific & Technical Pub.
12. Todd and Latham, 1999. The Mutation and Growth of Art by Computers. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc.
13. Sims, K., 1991. Artificial Evolution for Computer Graphics. *Computer Graphics*, **25**, No.4, 319-328.
14. Rowbottom, A., 1999. Evolutionary Art and Form. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., San Francisco, CA.
15. Bentley, P. & Wakefield, J., 1997. Conceptual Evolutionary Design by GAs. *Engineering Design and Automation Jnl* 3:2, John Wiley & Sons, Inc, 119-131.
16. Bentley, P. J. & Wakefield, J. P., 1997b. Generic Evolutionary Design. Chawdhry, P.K., Roy, R., & Pant, R.K. (eds) *Soft Computing in Engineering Design and Manufacturing*. Springer Verlag London Limited, Part 6, 289-298.
17. Thompson, A., 1995. Evolving Fault Tolerant Systems. *Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE Conf. Pub. 414, pp. 524-529.
18. Koza, J., 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.
19. Thompson, A. & Layzell, P., 1999. Analysis of Unconventional Evolved Electronics.

Communications of the ACM, April 1999 - Volume 42, Number 4, pp71-79.

20. Miller, J., Kalganova, T., Lipnitskaya, N., Job, D., 1999. The genetic algorithm as a discovery engine: strange circuits and new principles. To appear in Bentley and Corne (eds), *Creative Evolutionary Systems*, Morgan Kaufman Pub.
21. Koza, J.R. Bennett III, F. R., Andre, D. & Keane, M. A., 1999. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann Pub.
22. Funes, P. and Pollack, J., 1999. The Evolution of Buildable Objects. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc.
23. Sims, K., 1999. Evolving Three-Dimensional Morphology and Behaviour. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc.
24. Husbands, P., Jermy, G., McIlhagga, M., & Ives, R., 1996. Two Applications of Genetic Algorithms to Component Design. In *Selected Papers from AISB Workshop on Evolutionary Computing*. Fogarty, T. (ed.), Springer-Verlag, Lecture Notes in Computer Science, pp. 50-61.
25. Bentley, P. J., 1999b. Is Evolution Creative? In P. J. Bentley and D. Corne (Eds) Proceedings of the *AISB'99 Symposium on Creative Evolutionary Systems (CES)*. Published by The Society for the Study of Artificial Intelligence and Simulation of Behaviour (AISB), pp. 28-34.
26. Bentley, P. J., 1999c. An Introduction to Evolutionary Design by Computers. Chapter 1 in Bentley, P. J. (Ed.). *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., San Francisco, CA, 1-73.
27. Harvey, I., 1997. Cognition is not Computation: Evolution is not Optimisation. In *Artificial Neural Networks - ICANN97*, Gerstner, Germond, Hasler, and Nicoud (eds).
28. Vavak, F., & Fogarty, T., 1996. Comparison of Steady State and Generational GAs for Use in Nonstationary Environments. *Proceedings of the IEEE 3rd International Conference on Evolutionary Computation ICEC'96*, published by IEEE.
29. Page, J., Poli, R. and Langdon, W., 1999. Smooth Uniform Crossover with Smooth Point Mutation in Genetic Programming: A Preliminary Study, In R. Poli, P. Nordin, W. B. Langdon and T. Fogarty (Eds.), *Proceedings of the Second European Workshop on Genetic Programming - EuroGP'99*, Goteborg, May 26-27, 1999, Springer-Verlag.
30. Bentley, P. J. & Wakefield, J. P., 1996. Hierarchical Crossover in Genetic Algorithms. In *Proceedings of the 1st On-line Workshop on Soft Computing (WSC1)*, (pp. 37-42), Nagoya University, Japan.
31. Goldberg, D., 1999. The Race, the Hurdle, and the Sweet Spot: Lessons from Genetic Algorithms for the Automation of Design Innovation and Creativity. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc.
32. Bentley, P. J. and Kumar, S., 1999. Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pp.35-43.
33. Jakobi, N., 1996. Harnessing Morphogenesis. In *Proceedings of the international Conference on information Processing in Cell and Tissue*.
34. Taura, T. and Nagasaka,, 1999. Adaptive growth type representation for 3D configuration design. In Bentley, P.J. (Guest Ed.) *First Special Issue on Evolutionary Design, Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (AIEDAM) v13:3, Cambridge University Press, 171-184.
35. Hancock, P. and Frowd, C., 1999. Evolutionary Generation of faces. In Bentley, P. J. & Corne, D. W. (Eds) *Proceedings of the AISB'99 Symposium on Creative Evolutionary Systems (CES)*. Published by AISB, Sussex, UK. ISBN 1 902956 03 6.
36. Goldberg, D. E., 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley.