University of Huddersfield

University of Huddersfield Repository

Bentley, Peter J. and Wakefield, Jonathan P.

Generic evolutionary design

Original Citation

Bentley, Peter J. and Wakefield, Jonathan P. (1998) Generic evolutionary design. In: Soft Computing in Engineering Design and Manufacturing. Springer-Verlag, pp. 289-298. ISBN 3-540-76214-0

This version is available at http://eprints.hud.ac.uk/id/eprint/4053/

The University Repository is a digital collection of the research output of the University, available on Open Access. Copyright and Moral Rights for the items on this site are retained by the individual author and/or other copyright owners. Users may access full items free of charge; copies of full text items generally can be reproduced, displayed or performed and given to third parties in any format or medium for personal research or study, educational or not-for-profit purposes without prior permission or charge, provided:

- The authors, title and full bibliographic details is credited in any copy;
- A hyperlink and/or URL is included for the original metadata page; and
- The content is not changed in any way.

For more information, including our policy and submission procedure, please contact the Repository Team at: E.mailbox@hud.ac.uk.

http://eprints.hud.ac.uk/

Generic Evolutionary Design

P. J. Bentley[1] **and J. P. Wakefield**[2]

[1]*Department of Computer Science, University College London,*
Gower St., London WC1E 6BT, UK.
Tel. 0171 391 1329 P.Bentley@cs.ucl.ac.uk (corresponding author)
[2]*Division of Computing and Control Systems, School of Engineering*
University of Huddersfield, Huddersfield HD1 3DH, UK.
Tel. 01484 472107 J.P.Wakefield@hud.ac.uk

Keywords: generic evolutionary design, automated design, genetic algorithms

Abstract

Generic evolutionary design means the creation of a range of different designs by evolution. This paper introduces generic evolutionary design by a computer, describing a system capable of the evolution of a wide range of solid object designs from scratch, using a genetic algorithm.

The paper reviews relevant literature, and outlines a number of advances necessitated by the development of the system, including: a new generic representation of solid objects, a new multiobjective fitness ranking method, and variable-length chromosomes. A library of modular evaluation software is also described, which allows a user to define new design problems quickly and easily by picking combinations of modules to guide the evolution of designs.

Finally, the feasibility of generic evolutionary design by a computer is demonstrated by presenting the successful evolution of both conventional and unconventional designs for a range of different solid-object design tasks, e.g. tables, heatsinks, prisms, boat hulls, aerodynamic cars.

1. Introduction

Evolution is one of the most powerful search processes ever discovered [8]. In the natural world, evolution has created an unimaginably diverse range of designs of greater complexity than mankind could ever hope to achieve [6]. Natural evolution is the ultimate in generic evolutionary design.

In recent years, researchers have begun using computer algorithms that mimic this process of evolution, in order to automate stages of the human design process [10]. Designs have been successfully optimised using *genetic algorithms* (GAs), with some remarkable results [9]. More recently, simple conceptual designs have been generated by evolutionary computation [11]. However, as yet, evolution has not been used to perform the entire design process.

It is evident that natural evolution is eminently capable of generating new conceptual designs from scratch, evaluating these designs and optimising them. We are all living proof of this. Consequently, if evolutionary computation techniques of suitable similarity are used to perform the whole process of design, it seems probable that a computer system could become capable of equally creative and diverse design.

For this work it was decided that a computer system should be created that was capable of evolving the shape of a range of different solid-object designs from scratch. Such a system would demonstrate the feasibility of using evolution to perform the entire design process, and would show the creative potential of generic evolutionary design by a computer.

This paper briefly describes the investigation into and the creation of such a generic evolutionary design system. In addition, a range of designs successfully evolved by the system are presented and discussed.

2. Background

There are five separate but related areas of research relevant to the subject of generic evolutionary design by a computer:

(i) The optimisation of existing designs.
(ii) The generic optimisation of designs.
(iii) The creation of designs by computers.
(iv) The creation of art by computers.
(v) Genetic algorithms.

2.1. The Optimisation of Existing Designs

The development of non-generic optimisation systems, capable of optimising selected parts of existing designs, is a common area of research [10]. Numerous examples of the optimisation of designs exist, many using GAs or other adaptive search methods. For example, computers have been used to optimise: oil-pump pipelines, floorplans, structural topologies, finite impulse response digital filters, microwave absorbing materials, hydraulic networks, aircraft landing struts, VLSI layouts, and even spacecraft systems [8,9,10].

Some of these systems have been used with considerable success to optimise real-world problems. For example, as described by Holland, a design of a high-bypass jet engine turbine was typically optimised in eight weeks by an engineer; the genetic algorithm optimised a design in only two days, "with three times the improvements of the manual version" [9].

Whilst the wide variety of applications being tackled shows that computers can be used to successfully evaluate and optimise many different types of design, every one of these optimisation systems, without exception, suffers from two major drawbacks. Firstly, every one can only optimise existing designs - it would be quite impossible to use any of them to create a new design. Secondly, every one is application-specific - they can only optimise the single type of design they were created to optimise, and no others.

2.2. The Generic Optimisation of Designs

Generic design optimisation (i.e. the optimisation of more than one type of design by a single system) is a rare subject for research. As described in detail by Pham, Bouchard's 'Engineer's Associate' provides a limited generic framework to work with systems that can be represented by equations [11]. Alternatively, Culley's 'GPOS' (general purpose optimisation system) consists of a toolbox of optimisation algorithms, capable of optimising a range of different applications (once interfaced appropriately) [5].

However, Tong's 'Engineous' [14] is perhaps the most successful generic system, having been demonstrated on over twenty design optimisation tasks, including the optimisation of: turbine blades, cooling fans, DC motors, power supplies and a nuclear fuel lattice. A large portion of the system is comprised of complex interfacing software to allow the use of existing design evaluation packages. The system relies heavily on expert systems containing much application-specific knowledge to guide the evolution of a GA. Tong claims that "the current version of Engineous has demonstrated the profound impact such a system can have on productivity and performance" [14].

2.3. Creation of Designs by Computers

Research into the subject of design *creation* (typically concentrating only on conceptual designs) is growing. Early work concentrated on cognitive simulations, i.e. attempting to make a computer 'think' in the same way as a human, when designing. Such systems attempted to create descriptions of designs at an abstract level, typically using an expert system to 'design'. For example Dyer's 'EDISON' [7] represented simple mechanical devices such as doors and can-openers symbolically in terms of five components: parts, spatial relationships, connectivity, functionality and processes. A combination of planning and invention using 'generalisation', 'analogy' and 'mutation' attempted to modify these components to fulfil the design specification. Unfortunately, it seems that the abstract level at which reasoning was performed was too low, so the system was unable to handle any problems apart from the simplest cases [11]. Another approach consisted of invention based on 'visualising potential interactions' [15]. This generated descriptions of designs in terms of high-level components and the interactions between them, using qualitative reasoning and quantitative algebra. Again, the proposed system could only deal with highly simplified designs [11].

Many creative design systems reduce the complexity of the problem by presenting the computer with a number of high-level design building blocks which must be ordered correctly to form a design. For example, Pham describes a "preliminary design system" known as TRADES (TRAnsmission DESigner) [11]. When given the type of input (e.g. rotary motion) and the desired output (e.g. perpendicular linear motion), the system generates a suitable transmission system to convert the input into the output. This GA is presented with a set of building blocks such as rack and pinion, worm gear, and belt drive.

Perhaps the work that can most accurately be described as creative design is the recent work of Rosenman, who attempts to evolve new floorplans for houses [12]. Two dimensional plans are 'grown' using a simplified GA to modify 'cells' organised hierarchically using grammar rules. However, the system requires much problem-specific knowledge and the elaborate representation used may actually prevent complex shapes from being formed.

2.4. Creation of Art by Computers

The use of computers to create art (usually with GAs and similar adaptive search algorithms) is growing in popularity amongst some artists. For example, Todd and Latham have successfully evolved many three dimensional 'artistic' images and animations [13]. Their two-part system, consisting of 'Form Grow' and 'Mutator' uses an evolutionary strategy which creates and modifies shapes composed of 'artistic' primitive shapes (e.g. spiral, sphere, torus). John Mount showed his 'Interactive Genetic Art' on the internet (at http://robocop.modmath.cs.cmu.edu.8001/). This work utilised a GA to modify fractal equations that defined two dimensional images. Additionally, the biologist Richard Dawkins has demonstrated the ability of computers to evolve shapes resembling those found in nature [6]. Using a simple evolutionary strategy that modifies shapes arranged in tree-structures, he has produced images resembling the shapes of life-forms, e.g. 'spiders', 'beetles', and 'flowers'.

However, all of these art creation systems require the images being evolved to be evaluated by a human (i.e. artificial selection). Moreover, despite the fact that some of these systems can produce some complex three dimensional shapes [13], none of them have been used to produce anything more than 'pretty pictures'.

The system described in this paper combines for the first time the creative evolutionary techniques pioneered by artists (and biologists) with the more rigorous methods of automatic creative design. This has resulted in a novel generic design system which has the 'creative properties' of the art systems and is capable of the generation of a wide range of useful designs [1]. Furthermore, it is the 'innovative flair' (Goldberg, 1989) of the genetic algorithm that gives the system such capabilities.

2.5. The Genetic Algorithm

Perhaps uniquely for one type of search algorithm, the genetic algorithm has become widely used in all of the areas of research related to generic evolutionary design. In these and many other domains, the GA has been shown repeatedly to be a highly flexible algorithm, capable of finding good solutions to a wide variety of problems [8].

The GA is based upon the process of evolution in nature. Evolution acts through large populations of creatures which individually reproduce to generate new offspring that inherit some features of their parents (because of random *crossover* in the inherited chromosomes) and have some entirely new features (because of random *mutation*). Natural selection (the weakest creatures die, or at least do not reproduce as successfully as the stronger creatures) ensures that more successful creatures are generated each generation than less successful ones. In nature, evolution has produced some astonishingly varied, yet highly successful forms of life. These creatures can be thought of as good 'solutions' to the problem of life. In other words, evolution optimises creatures for the problem of life.

In the same way, within a genetic algorithm a population of solutions to the problem is maintained, with the 'fittest' solutions (those that solve the problem best) being randomly picked for 'reproduction' every generation. 'Offspring' are then generated from these fit parents using random crossover and mutation operators, resulting in a new population of fitter solutions [8]. As in nature, the GA manipulates a coded form of the parameters to be optimised, known as the *genotype*. When decoded, a genotype corresponds to a solution to the problem, known as a *phenotype*.

The *robustness* of GAs [9], combined with the fact that the human design process has been formally compared to the working of a GA, and that the GA is the closest analogy to natural evolution in Computer Science, make the choice of a GA in a generic evolutionary design system seem wholly justified [1].

3. The Generic Evolutionary Design System

When applying a genetic algorithm to any new application, four main elements must be considered. First, the phenotype must be specified, i.e. the allowable solutions to the problem must be defined by the specification and enumeration of a search space. Second, the genotype (or coding of the allowable solutions) must be defined. Third, the type of genetic algorithm most suitable for the problem must be determined. Fourth, the fitness function must be created, in order to allow the evaluation of potential solutions of the problem for the GA.

Since a genetic algorithm was used to form the core of the generic evolutionary design system, these four elements can be identified in the system. Designs are searched for using a multiobjective genetic algorithm as the 'search-engine' to evolve solutions. To achieve this, the GA manipulates hierarchically organised genotypes (or coded solutions). The genotypes are mapped to phenotypes (or designs) defined by a low-parameter spatial-partitioning representation. These phenotypes are analysed by modular evaluation software, which provides the GA with multiple fitness values for each design. Figure 1 illustrates how these four elements are combined to allow the evolution of a range of different solid object designs from scratch.

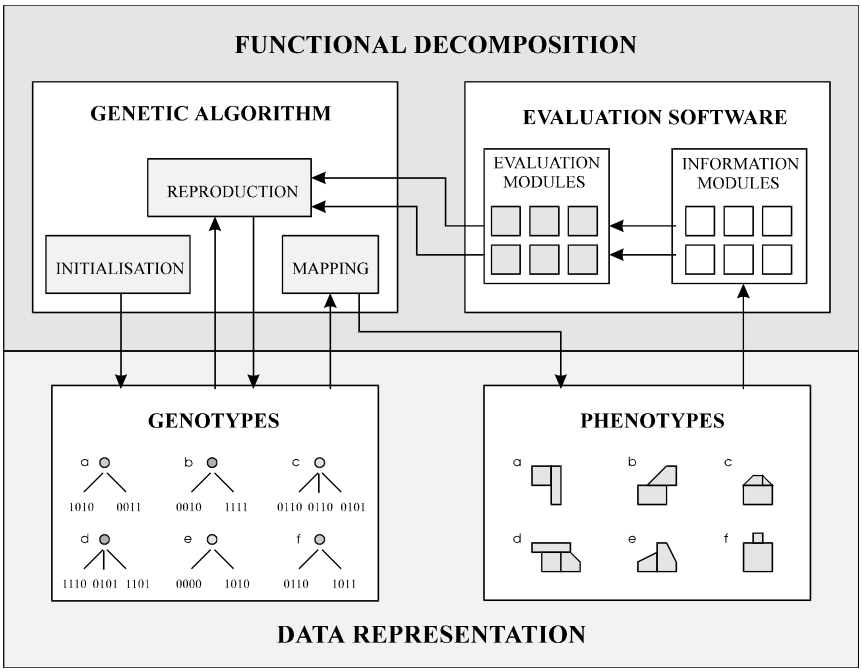

Figure 1. Block diagram of the generic evolutionary design system.

3.1. Phenotypes

Evolving designs, or phenotypes, from scratch rather than optimising existing designs requires a very different approach to the representation of designs. When optimising an existing design, only selected parameters need have their values optimised (e.g. for a jet-turbine blade, such parameters could define the length and cross-sectional area at specific parts of the blade). To allow a GA to create a new design, the GA must be able to modify more than a small selected part of that design - it must be able to modify every part of the design. This means that a design representation is required, which is suitable for manipulation by GAs. Many possible representations exist, and some have been used by the evolutionary-art systems: Todd and Latham [13] used a variant of constructive solid geometry (CSG), others have used fractal equations (e.g. John Mount), and Dawkins used tree-like structures [6]. However, for a system capable of designing a wide variety of different solid object designs, a more generic representation is needed.

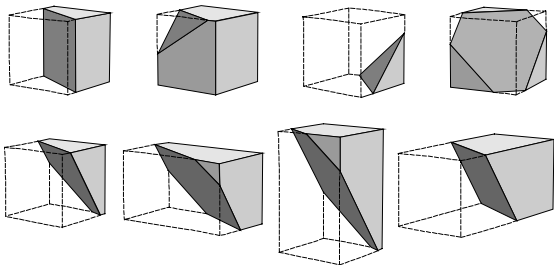

Figure 2. Examples of primitive shapes used to represent designs.

After some investigation, a new variant of spatial-partitioning representation (known as 'Clipped Stretched Cuboids'), was developed for this work. This representation combines methods from CSG and traditional spatial partitioning representations, to allow the definition of a wide range of solid objects using a number of primitive shapes in combination [3]. Primitive shapes consist of a rectangular block or cuboid with variable width, height and depth, and variable three dimensional position. Every cuboid can also be intersected by a plane of variable orientation (see fig. 2), to allow the approximation of curved surfaces. Intersected cuboids, or primitives, require nine parameters to fully define their geometry. Designs are defined by a number of non-overlapping primitives.

This solid-object representation is capable of the definition of a wide range of solid objects using relatively few primitives to partition the space. Significantly, the fewer the primitives in a design, the fewer the number of parameters that need to be considered by the GA. Additionally, this representation enumerates the search-space such that similar designs are placed close to each other, minimising discontinuities, and easing the task of finding an evolutionary path from a poor design to a better design [1].

3.2. Genotypes

The genetic algorithm within the system never directly manipulates phenotypes. Only coded designs, or genotypes are actually modified by the genetic operators of the GA. Every genotype consists of a single chromosome arranged in a hierarchy consisting of multiple blocks of nine genes, each gene being defined by sixteen bits, see fig. 3.4. This arrangement corresponds to the spatial partitioning representation used to define the phenotypes, with each block of genes being a coded primitive shape and each gene being a coded parameter.

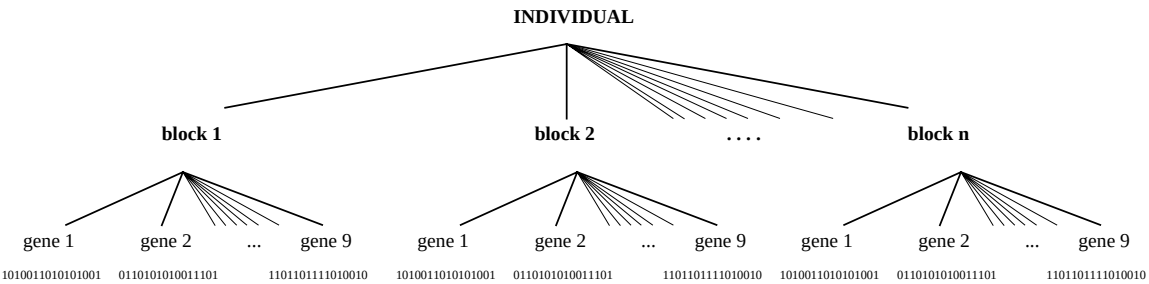

Figure 3. Hierarchically structured genotype of a design

A mutation operator is used within the genetic algorithm to vary the number of primitives in a design by adding or removing new blocks of nine genes from chromosomes. This permits evolution to optimise the number of primitives in addition to the geometries of primitives in designs. (A new primitive is added to a design by splitting a randomly chosen primitive into two. A primitive is removed by simply deleting that primitive from the genotype). However, varying the length of chromosomes in this way can cause the crossover operator to produce meaningless offspring [4]. To overcome this, a new type of crossover operator, known as hierarchical crossover, was developed. This new version of crossover uses the hierarchical arrangement of the chromosomes to find points of similarity between two chromosomes of different sizes. Once such points are found, hierarchical crossover uses the tree-structure of the chromosomes to efficiently generate new offspring without loss of meaning [4].

Hierarchical crossover is used by the GA to generate all offspring (i.e., with a probability of 1.0). Mutation is used to vary the number of primitives in a design with a default probability of 0.01 and a standard mutation operator is used to vary single bits within genes with a default probability of 0.001.

3.3. Genetic Algorithm

The genetic algorithm used within the system is more advanced than Goldberg's simple GA [8]. For example, two populations of solutions are maintained: the main *external population*, and the smaller *internal population*. All new solutions are held in the internal population where they are evaluated. They are then moved into the external population (i.e. 'born' into the 'real world'), replacing only the weakest members of the external population. Other different features include the use of an explicit mapping stage between genotypes and phenotypes, and the use of multiobjective techniques within the GA.

To begin with, the GA has the internal population of solutions initialised with random values to allow the evolution of designs from scratch (i.e., the GA begins with randomly shaped 'blobs'). However, if required, a combination of random values and user-specified values can be used to allow the evolution of pre-defined components of designs, or of selected parts of designs.

The GA then uses an explicit mapping stage to map the genotypes to the phenotypes. This resembles nature, i.e., the DNA of an organism is never 'evaluated' directly; first the phenotype must be grown from the 'instructions' given in the DNA, then the phenotype is evaluated [6]. By performing this process explicitly, the system is able to gain some advantages. For example, should a symmetrical design be required, only half a design needs to be coded in the genotype and hence evolved by the GA. This partial design can then be reflected during the mapping stage to form a complete design, which is then evaluated. This mapping stage is also used to enforce the rules of the solid-object representation, by ensuring that any designs with overlapping primitives are corrected so that their primitives touch rather than overlap [2].

Next, the GA calls user-specified modules of evaluation software to analyse the phenotypes and obtain multiple fitness values for each individual solution (most design problems are multiobjective problems). The GA must then determine from these multiple fitness values which phenotypes are fitter overall than others. In other words, the GA has to be able to place the phenotypes into order of overall fitness, using multiobjective optimisation techniques to handle the many separate fitness values produced by the evaluation software.

After performing comparisons between the performances of existing and new multiobjective ranking techniques, it was found that one of the new methods developed for this work allowed the GA to evolve the best designs most consistently. This multiobjective method automatically scales the separate fitness values of each phenotype, according to the effective ranges of the corresponding functions, in order to make them commensurable [1]. The fitness scores are then simply summed to provide a single, overall fitness value for each phenotype. In addition, by multiplying each scaled fitness value by a user-defined weighting value before aggregation, the new method also incorporates the concept of 'importance', allowing a user to increase or decrease the relative importance of any objective [1].

Once overall fitness values have been calculated for each individual solution, the GA moves the individuals from the internal population where all new individuals are held, into the main external population. However, unlike the simple GA, this GA does not replace an entire population of individuals with new individuals every generation. In a similar way to the steady-state GA, this GA only replaces the weakest (less fit) individuals in the external population with new individuals from the smaller internal population, allowing the fittest individuals to remain in the external population over multiple generations. Unusually, the GA also prevents very fit individuals from becoming immortal by giving every individual in the external population a pre-defined lifespan. Once the individual reaches this lifespan, they become very unfit and thus are quickly 'killed' by new individuals taking their places. This prevents poor individuals with high scores, caused by the random variations of noisy evaluation functions, from corrupting evolution [1].

Finally, the GA favours individuals with higher overall fitnesses when picking 'parents' from the external population. The randomly chosen parent solutions (with fitter solutions preferentially selected) are then used to generate a new internal population of offspring using hierarchical crossover and the mutation operators.

The GA then maps the new genotypes to the phenotypes, evaluates the new phenotypes, and continues the same process as before. This iterative process continues until either a specified number of generations (i.e. loops) have passed, or until an acceptable solution has emerged.

3.4. Evaluation Software

All parts of the system described so far are generic, i.e. they can be applied to a wide range of different solid-object design problems. However, there is an element of the system that must inevitably be specific to individual design applications: the evaluation software. Designs must be evaluated to instruct the GA how fit they are, i.e.

how well they perform the desired function described in the design specification. Hence, the evaluation software is a software version of the design specification, which must be changed for every new design task.

In an attempt to reduce the time needed to create evaluation software for a new design problem, all parts of the various different types of evaluation software created for this work have been implemented as re-usable modules. In other words, it is proposed that many designs can be specified by using a number of existing evaluation modules in combination. Moreover, wholly new design tasks will only require the creation of modules of evaluation software that do not already exist, thus dramatically shortening the time needed to apply the system to a new application. Over time a large library of such modules could be developed, to reduce the future need for new modules. Examples of the existing modules in the library of evaluation software developed as part of this project include: *minimum size*, *maximum size*, *specific mass*, *specific surface area*, *stability*, *supportiveness*, *ray-tracing*, and *particle-flow simulator*.

In addition to a library of different evaluation software modules (or fitness functions), a library of phenotype information modules is maintained. This is necessary because many modules of evaluation software require specific information about a design in order to calculate how fit that design is. Using a distinct information module to calculate, say, the mass of a design, allows all evaluation modules that need this value to share the information generated. Hence, such information on phenotypes need only be generated once, to supply all evaluation modules that require it. Examples of the information modules in the library developed as part of this project include: *vertices*, *mass*, *centre of mass*, *extents*, *primitive extents* and *surface area*.

Consequently, complete design applications are specified to the evolutionary design system by the selection of a combination of modules of evaluation software, and their corresponding desired parameter values. The system then enables the appropriate information modules which supply all of the evaluation modules with the necessary information on the current phenotype. A number of separate fitness values are generated by the evaluation modules for each design, which are used by the GA to guide evolution to good solutions.

4. Designs Evolved by the System

In total, fifteen different design tasks were presented to the system: tables, sets of steps, heatsinks, optical prisms (right-angle, roof, derotating, rhomboid, penta, abbe, porro), and streamlined designs (train fronts, boat bows, boat hulls, saloon cars, sports cars). Each of these tasks involved the evolution of a design with an entirely different shape, in order to allow that design to perform the desired function. Despite some of these problems being deceptive for the GA, this generic system was able to evolve not only fit, but acceptable designs (as judged by humans) for all fifteen problems [1]. Most designs took around 500 generations to evolve, using internal and external population sizes of 160 and 200 respectively.

The first task was to evolve the design of a table. This was specified by using five evaluation modules: *size*, *mass*, *flat upper surface*, *supportiveness* and *unfragmented.* These defined that good table designs should be an appropriate size and mass, should have a flat upper surface capable of supporting heavy objects without the table toppling over, and that the design should be whole (i.e. no part should 'float free' of the main design). Figure 4 shows an evolved design which uses four legs to provide the required stability. Alternative solutions have used other concepts such as a single wide base or a very low centre of mass to provide stability [2].

The second task was to evolve the design of a small set of steps, specified with similar modules of evaluation software as used for the table problem, except that three flat surfaces at specified heights were desired. Figure 5 shows an intricate evolved design using two side supports for stability, with the top step being further supported by a column at the rear. Behind the steps the design is hollow to reduce the mass.

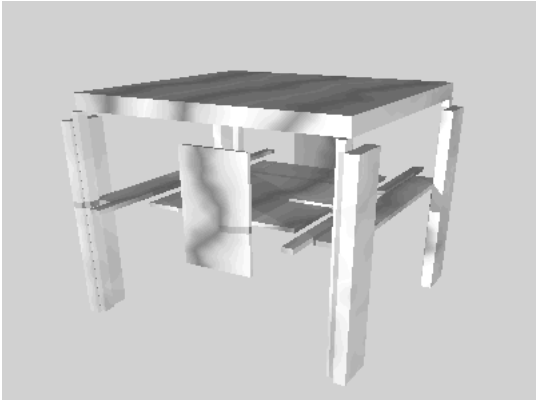

Figure 4 Evolved Table

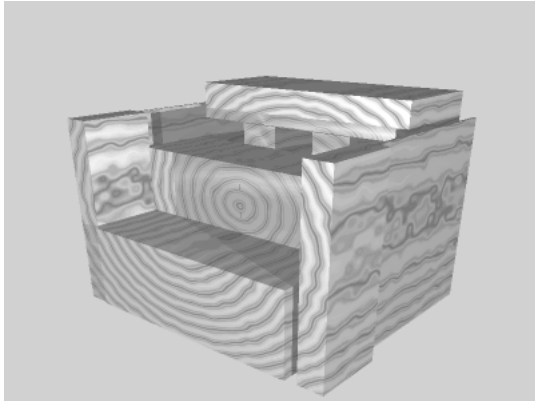

Figure 5. Evolved set of steps

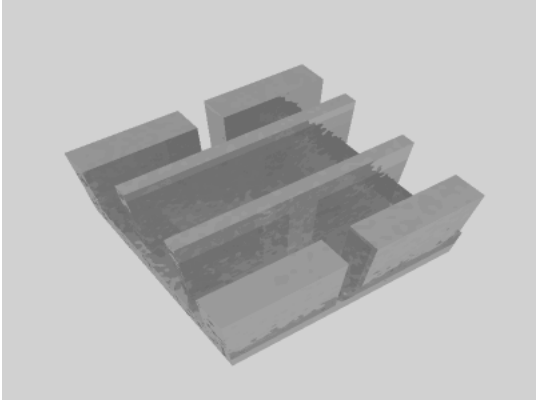

Figure 6 Evolved heatsink

Figure 7. Evolved porro prism

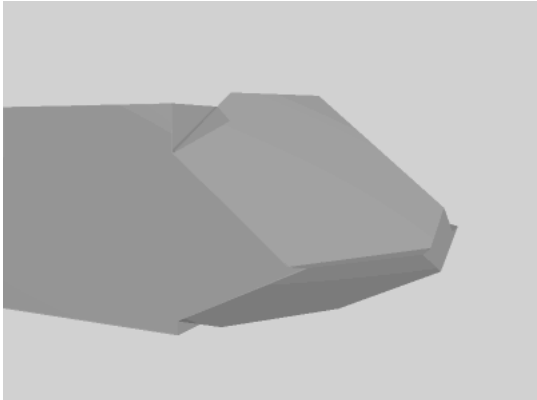

Figure 8. Side view of evolved front of a train

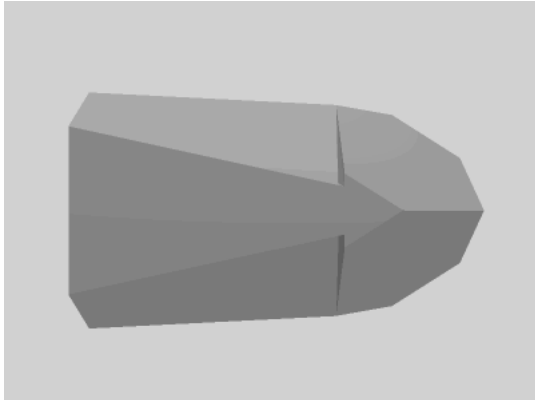

Figure 9. Underside of evolved boat hull

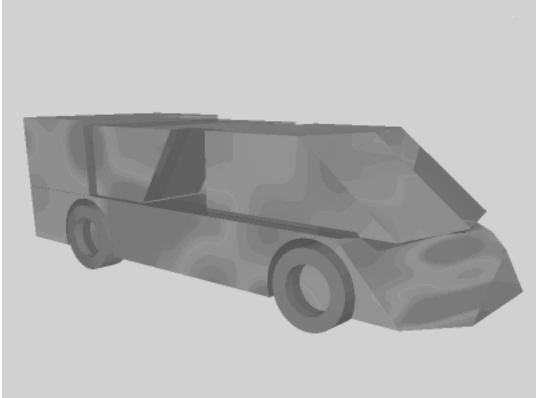

Figure 10. Evolved saloon car (wheels added)

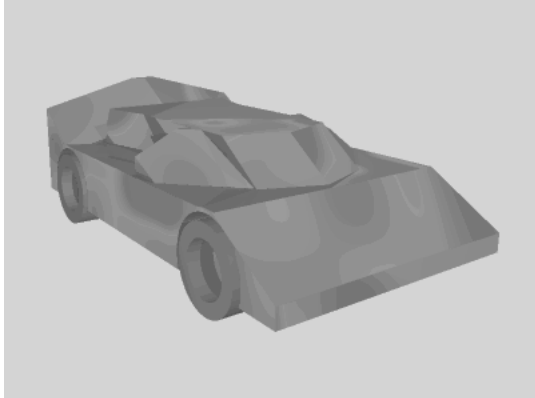

Figure 11. Evolved sports car (wheels added)

The third task was to evolve designs of heatsinks (to dissipate the heat of CPUs). This task was specified using the evaluation modules: *size*, *mass*, *unfragmented*, and *surface area*. A very high value for the surface area was desired (to define, in effect, that the surface area, and hence the approximate ability of the heatsink to radiate heat, should be maximized). A number of unusual designs were evolved, with the system often dramatically increasing the number of primitives used to represent each design in order to create detailed and uneven surfaces with large surface areas. Figure 6 shows a more traditional-looking evolved heatsink design.

The fourth type of problem was to evolve a number of different types of optical prism. All of these problems were specified using the evaluation modules: *size*, *unfragmented*, and *ray-tracing*. The ray-tracing module was used to define the characteristics of light travelling into the prisms, and evaluate how well the characteristics of the emerging light matched the desired characteristics for each type of prism. Figure 7 shows an almost perfect porro prism evolved by the system (for this problem using a collection of initially randomly-positioned, previously-evolved right-angle prisms).

The fifth type of problem was to evolve a number of different types of 'streamlined' designs. These were specified using the evaluation modules *size*, *unfragmented*, and *particle-flow simulator*. The 'particle-flow simulator' was used to provide an approximation of water and air-flow by firing particles at designs and calculating the forces generated when collisions with the designs occurred. By defining that minimal forces on the front of designs were required, designs with low water or air-resistance were specified. Figures 8, 9, 10, and 11 show successfully evolved 'streamlined' designs of a train, boat hull, saloon car and sports car, respectively. As can be clearly seen, the system has independently discovered a variety of techniques to allow the desired forces to be generated on each design. For example, fig. 8 shows a design of a train with a pointed nose, shaped to minimise wind resistance and generate an overall down-force. Figure 9 shows the underside of a boat hull, angled to cut through water cleanly and provide the required amount of up-force. Figure 10 shows a design of a saloon car, evolved around a fixed chassis, which uses a sloping windscreen and bonnet (windshield and hood) to reduce wind resistance and generate the desired down-force. Finally, fig. 11 shows an evolved sports car that has a sloped bonnet, curved windscreen and large back spoiler which all work in unison to generate the required amount of force pushing down on each wheel, whilst minimising the air-resistance.

5. Conclusions

This paper has presented a new way of using computers in design. It has shown that it is possible, feasible and useful to produce a generic evolutionary design system capable of successfully creating a range of new and original solid-object designs.

This novel computer system has four main components:

- A new low-parameter spatial-partitioning representation, used to define the shape of solid-object designs.
- Hierarchically structured genotypes combined with a new hierarchical crossover operator, which allow child designs to be efficiently generated from parent designs with different sized genotypes without loss of meaning.
- A steady-state multiobjective genetic algorithm, using an explicit mapping stage between genotypes and phenotypes, preferential selection of parents and a life-span operator, which forms the main search-engine at the core of the system.
- Modular evaluation software, which is used to guide evolution to functionally acceptable designs, with new design tasks being quickly specified by the user picking combinations of existing evaluation modules from a library.

The research described in this paper has demonstrated the use of a computer to perform generic evolutionary design by evolving consistently acceptable designs for fifteen different design tasks. Designs evolved by the system were based on sound conceptual ideas, 'discovered' independently by the system. The shapes of designs were optimised in order to ensure that they performed the desired function accurately. The system evolved a range of conventional and unconventional designs for all problems presented to it.

In conclusion, evolutionary design has been performed in nature for millennia. This research has made the first steps towards harnessing the power of natural evolutionary design, by demonstrating that it is possible to use a genetic algorithm to evolve designs from scratch, such that they are optimised to perform a desired function, without any human intervention.

References

[1] Bentley, P. J., 1996, *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*. Ph.D. Thesis, University of Huddersfield, Huddersfield, UK.

[2] Bentley, P. J. & Wakefield, J. P., 1996a, The Evolution of Solid Object Designs using Genetic Algorithms. *Modern Heuristic Search Methods*, John Wiley & Sons Inc., **Ch 12**, **197-211**.

[3] Bentley, P. J. & Wakefield, J. P., 1996b, Generic Representation of Solid Geometry for Genetic Search. *Microcomputers in Civil Engineering* **11:3**, **153-161**.

[4] Bentley, P. J. & Wakefield, J. P., 1996c, Hierarchical Crossover in Genetic Algorithms. *Proceedings of the 1st On-line Workshop on Soft Computing* (WSC1), Nagoya University, Japan, **37-42**.

[5] Culley, S. J. and Wallace, A. P., 1994, Optimum Design of Assemblies with Standard Components. *Proc. of Adaptive Computing in Engineering Design and Control - '94*, Plymouth, **163-168**.

[6] Dawkins, R. 1986, *The Blind Watchmaker*, Longman Scientific & Technical Pub.

[7] Dyer, M. Flower, M. and Hodges, J., 1986, 'EDISON': an engineering design invention system operating naively. *Artificial Intelligence* **1**, **36-44**.

[8] Goldberg, D. E., 1989, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison-Wesley.

[9] Holland, J. H., 1992, Genetic Algorithms. *Scientific American*, **66-72**.

[10] Parmee, I C & Denham, M J, 1994, The Integration of Adaptive Search Techniques with Current Engineering Design Practice. *Proc. of Adaptive Computing in Engineering Design and Control - '94*, Plymouth, **1-13**.

[11] Pham, D. T. & Yang, Y., 1993, A genetic algorithm based preliminary design system. *Journal of Automobile Engineers* **v207:D2**, **127-133**.

[12] Rosenman, M. A., 1996, A Growth Model for Form Generation Using a Hierarchical Evolutionary Approach. *Microcomputers in Civil Engineering* **11:3**, **163-174**.

[13] Todd, S. & Latham, W., 1992, *Evolutionary Art and Computers*, Academic Press.

[14] Tong, S.S., 1992, Integration of symbolic and numerical methods for optimizing complex engineering systems. *IFIP Transactions (Computer Science and Technology)* **vA-2**, **3-20**.

[15] Williams, B. C., 1990, Visualising potential interactions: constructing novel devices from first priciples. *Proc. of Eighth National Conference on Artificial Intelligence (AAAI-90)*, Boston, Mass.