New Trends in Evolutionary Computation

P. J. Bentley, T. G. W. Gordon, J. Kim and S. Kumar
Department of Computer Science
University College London, Gower St., London, UK
P.Bentley@cs.ucl.ac.uk

Abstract- In the last five years, the field of evolutionary computation (EC) has seen a resurgence of new ideas, many stemming from new biological inspirations. This paper outlines four of these new branches of research: Creative Evolutionary Systems, Computational Embryology, Evolvable Hardware and Artificial Immune Systems, showing how they aim to extend the capabilities of EC. Recent, unpublished results by researchers in each area at the Department of Computer Science, UCL are provided.

1 Introduction

This paper describes four recent branches in evolutionary computation (EC) currently under investigation at University College London (UCL) that take us a few steps closer to the target of a mature technology based on evolutionary processes. We show how these new ideas are based on a better understanding of the capabilities of evolutionary search. Instead of forcing evolution to do what we think it should by adding heuristics or local searches, we are harnessing the innate abilities of evolution for exploration, innovation, adaptability and distributed search, often identified through new biological inspirations (Bentley, 2001). Examples and results are provided by researchers from each area, based at the Department of Computer Science, UCL.

We begin by examining how evolutionary algorithms are being extended and employed for unconventional applications, enabling the development of *creative evolutionary systems.* We then show how new advances in evolutionary design of electronic circuits are possible, using similar ideas in *evolvable hardware*. Next, we describe how *computational embryology* is opening up new possibilities of scalability and adaptability. Finally, we present one of the latest major types of evolutionary algorithm: an *artifical immune system* and show its abilities for machine learning. The paper concludes with a brief summary and discussion.

2 Creative Computation

The techniques of evolutionary computation have long been used for optimisation, with some considerable success. But some researchers are beginning to argue that evolution is not a natural optimiser. In nature, evolution propagates change through populations that exist in dynamic, interacting and ever-changing environments. Concepts of optima are meaningless in such natural systems – at most we can talk of 'better' or 'worse' solutions to the day-to-day problems faced by organisms.

Evolution works very well with such undefined and changing problems. It may not be the best optimiser, but when optima are irrelevant, its abilities to explore possibilities and find creative (if transitory) solutions seems unsurpassed (Bentley and Corne, 2001).

Recent research is focussing on these abilties of evolution. Rather than attempt to optimise static and simplified functions, embedding knowledge (and constraints) into the search, new work applies evolutionary algorithms to dynamic and ill-defined problems. Such research 'traditionally' took place only in artificial life, but researchers now investigate how evolution can generate novelty for unconventional applications such as music composition, art, conceptual design and even novel fighter pilot strategies (Bentley and Corne, 2001).

Previous and forthcoming work (Bentley, 2000; Bentley and Corne, 2001) has explored the huge variety of new, creative applications being tackled by evolutionary practitioners today[1]. But how do we enable evolution to handle creativity? One common definition of creativity is *removal of constraints* (see Bentley (1999a) for a review of this and other definitions). It is no coincidence, then, that when we examine how creative evolutionary systems differ from more traditional evolutionary systems, we see that constraints of one form or another have been relaxed or removed. Some researchers do this quite explicitly – for example the recent work of Ian Parmee concentrates on the development of interactive evolutionary systems that allow users to relax functional constraints for engineering design problems (Parmee & Bonham, 2000). But there are other, important constraints that must be considered if we are to enable creative evolutionary systems to generate novelty.

Traditional implementations of evolutionary search suffer from the same fundamental drawbacks as all conventional search algorithms. They rely on a good

[1] Here we mean new to evolutionary computation – other AI approaches have been used to tackle creative applications in the past.

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. Evolution, like all search algorithms, is limited and constrained by the representation it can modify. While relaxing functional and parameter constraints will permit evolution to arrive at a larger diversity of solutions, only by modification of the representation can we enable evolution to innovate. It seems that the remarkable results obtained by creative evolutionary systems require the removal of constraints within the representations, and not only the fitness functions. And by using a different type of representation, we can cause this to happen automatically.

When the parameters of our representation do not define the solution directly, when they define a set of components from which the solution is constructed, we permit far greater freedom for evolutionary search (Bentley, 2000). 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.

But while component-based representations enable evolution to discover novel solutions, they do not allow us tackle unevaluatable or ill-defined problems. To do this we often need to add human interaction to the evolutionary process. Our judgement must contribute to or replace the fitness functions (or any other part of the evolutionary algorithm that helps generate selection pressure). When we modify EAs in this way, we call them interactive evolutionary algorithms, or *collaborative* evolutionary algorithms (Bentley and Corne, 2001).

There are a large number of different approaches used in this area. For example, Furuta et al (1995) allowed users to judge the aesthetics of evolving bridges in addition to their structural evaluation, and employed 'psychovectors' to quantify aesthetic factors. Others use multiobjective optimisation methods to combine user input and fitness functions (Bentley, 1999b), some use fuzzy logic to aid the input of knowledge into the search (Parmee & Bonham, 2000). Most evolutionary art systems will present users with some or all of the evolving population, and allow them to rank or vote on the quality of the images (Rowbottom, 1999). Some, like the system described in Adrian Thompson's recent work (Ch. 9 in Bentley and Corne, 2001), even allow users to 'vote with their feet' and physically move themselves towards evolving solutions that they prefer.

2.1 Interactive Genetic Algorithm Designer

The addition of user input into an evolutionary algorithm is so easy, and the results often so good, that it is surprising how few researchers actually permit it. To provide an example of such a system in operation, Bentley's *genetic algorithm designer*, GADES (Bentley, 1999b) was modified.

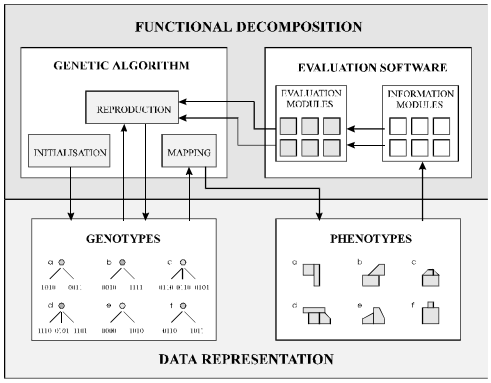
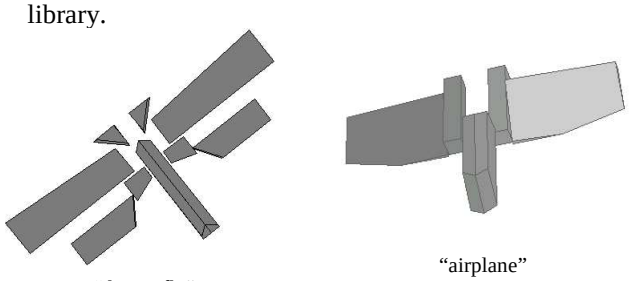


Figure 1 The generic evolutionary design system.

The system comprises four main elements (fig. 1):

- A low-parameter spatial-partitioning representation, used to define the shape of solid-object designs by shaping and combining separate solid 'blocks'.
- Hierarchically structured genotypes combined with a 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.



"dragonfly"
evolved with human guidance only

"airplane"
evolved using human guidance and objective functions

Figure 2 Shapes evolved by the interactive GADES.

A new evaluation module was added which simply displayed a 3D rendered image of each member of the evolving population to the user in turn, and asked for a fitness score to be input (a number between 0 and 9). With a reduced population size of ten individuals, slightly higher mutation rates than normal (to slow convergence) and

about 30 or 40 generations, a wide variety of 'interesting solutions' were evolved. Because of the time needed to judge the solutions, each took around half an hour. Figure 2 shows how creative the results of evolution were, when combined with human guidance. The left image illustrates a shape evolved with human guidance alone, the right image shows a shape evolved with human guidance as one objective, and two other objectives (to ensure the shape was not fragmented and also to reduce 'wind resistance' by ensuring a streamlined shape). Other results are provided in (Bentley and Corne, 2001).

3 Evolvable Hardware

There are of course more significant applications than evolving shapes that look like dragonflies or airplanes (however streamlined they may be). One important new area is *evolvable hardware*: the automatic design and synthesis of electronic circuits. The most exciting area within this field is that of intrinsic hardware evolution, where evolved designs are tested for fitness *in vivo* on field programmable gate arrays (FPGAs).

Modern circuits can contain a large number of components. Human designers need to reduce the search space of all functions of these components to a manageable size. To do this, they tend to work in a space of lower dimensionality in which they are expert. For instance, some designers treat all components as perfect digital gates, when in fact the components are physical devices that behave as high gain analogue amplifiers.

In order for this digital design abstraction to be realised in hardware, we have to restrict the temporal behaviour of the physical circuit, imposing set-up and hold times on sections of the circuit. This allows the analogue circuit to appear to behave as digital gates. Such restrictions not only prevent us from taking advantage of a huge amount of potentially useful gate behaviour, but also force us to spend a great deal of the design effort ensuring that the phase rules are adhered to throughout the circuit.

Evolutionary search works best without adding such constraints. Indeed, by allowing evolution to use the analogue behaviour abstracted away by digital designers, we can enable the generation of more efficient circuits, and perhaps even aid the evolutionary process through improved evolvability. Thompson's (1996) tone discriminator is an example of such work. Parts of this circuit behave digitally, as specified by the FPGA vendor. However, the overall behaviour is far from digital, resulting in highly original and non-human electronic circuit designs.

3.1 Evolving Adders

An intrinsic evolvable hardware platform has been developed to further research at UCL. We selected the Xilinx Virtex FPGA for our experiments[2].

Although the Virtex architecture is course-grained it can be configured using JBits (the Java configuration API from Xilinx) at a fine level of detail. However to allow the genetic algorithm to manipulate the configuration at such fine granularity that JBits allows, a directly mapped binary representation could not be easily used. To avoid unknown and unwanted biases in the encoding, each resource that could be modified by JBits was encoded as a separate integer gene. The exceptions to this were the lookup table (LUT) configuration, which were each sixteen bits.

Routing representation was also an issue. Each configurable logic block (the Virtex is arranged as an array of CLBs) is driven independently, so it is possible for contention to arise between two drivers if two output multiplexers from two different CLBs drive the same line. Previously reported intrinsic evolution using Virtex FPGAs has either worked with fixed routing (Hollingworth et al, 2000), or the method of contention avoidance has not been reported (Levi, 2000).

To avoid contention we modified the representation. It was noted that although CLB input multiplexers can connect to many nearest neighbour routing lines (singles), the connections between the output multiplexers and singles are sparse, and few can connect to any that their neighbours can. In fact, only eight of the possible forty-eight connections had to be prohibited to prevent any possible contention arising. Only singles were encoded on the chromosome, and connection points between singles were not evolved. Although the representation was necessarily restricted with the human design principle of contention avoidance, it allows the genetic algorithm access to manipulate almost all other configurable features on the FPGA. Searching this unconstrained design space offers the opportunity to find innovative circuits designed expressly for the Virtex architecture. An overview of the chromosome structure for one CLB and its associated routing is shown in table 1.

Number of Genes	Type of Gene	No. of Values
16	LUT Input MUXes	27
2	Clock Input MUXes	11
2	SR Input MUXes	10
2	Clock Enable Input MUXes	11
4	Other Input MUXes	4
64	LUT Configuration	2
48	Other CLB Logic	1-3
8	Output signal MUX	13
40	Output MUX to single switches	2

Table 1 Overview of the genotype for one CLB.

[2] Virtex 2.5V Field Programmable Gate Arrays Databook V2.4 available at: http://www.xilinx.com/partinfo/ds003.pdf.

A small rectangle of the chip was selected for evolution. The cells on the edges of this area only allowed to use the routing connections to the rest of the evolved area, and not outside. For instance the cell on the northeast corner was restricted to use only output multiplexer connections to single lines travelling south and west.

3.2 Experiments

A genetic algorithm was developed to evolve this structure. In addition to the behaviour of standard integer genetic algorithms, it was also required that the number of alleles that each gene can assume can be set independently of the others. We chose to validate this genetic algorithm by evolving in simulation a circuit representation that Miller et al (1997) had examined, the two-bit adder.

The circuit is represented by an indexed rectangle of cells. They are indexed from the top left cell, row-wise then column-wise. Each cell has two inputs and one logic function. The function may either be a two input logic gate or a two data input multiplexer. Inputs to a cell can be either from other cells, or the circuit inputs. The circuit inputs are the test input vector, the inverted test input vector, logic 0 or logic 1. To avoid feedback, each input must be from a cell with a lower number than the cell itself. The circuit outputs required are restricted to the top and right hand side of the cell array.

The circuit is encoded as an integer string. There is a triplet of integers for each cell, representing the sources of the two cell inputs and the cell function, with the triplet locus mapping to the cell index. If the cell function is a logic gate, the function allele represents a specific logic gate. If the cell function is a multiplexer, then the allele represents the multiplexer control signal source, which can be either the output of another cell or a circuit input. Table 2 shows the list of functions used. Cell outputs are represented by an integer each, the allele referring to the output cell index.

Fitness was measured by subjecting each candidate circuit to a test vector containing the complete two-bit adder truth table. One fitness point was awarded for each correct input/output sequence, giving a total of 96 for maximum fitness. The circuit inputs were therefore A0, A1, B0, B1, Carry In, !A0, !A1, !B0, !B1, !Carry In, 0, and 1. The outputs were S0, S1, and Carry Out.

Allele	Function	Allele	Function
0	A . B	7	!A
1	A . !B	8	!A +B
2	!A . B	9	!B
3	A ⊕ B	10	!A + B
4	A + B	11	!A + !B
5	!A . !B	12....	!C . A + C . B, C = circuit input 0
6	!A ⊕ B	...(n)	!C . A + C . B, C = input (n-12)

Table 2 List of function to allele mappings used.

Uniform crossover was used with two-member tournament selection (the winner of each tournament was selected with 70% probability). The mutation rate was set to 5% of all genes in the population. The population size was set to 30. 20,000 generations (unlike Miller et al's 40,000-80,000) seemed sufficient to produce good results.

3.3 Results

10 runs were each made for cell array sizes of 3x3, 3x4, and 4x4. Table 3 shows overall results. Fitnesses and deviations have been scaled to 100.

Array Size	Mean fitness of Best Solutions	Std. Dev. Of Best Solutions	% of Runs with Perfect Solutions
3x3	96.25	4.99	50%
4x3	96.88	3.71	50%
4x4	97.50	4.03	60%

Table 3 Results from 10 runs of 2 bit adder evolution across a range of array sizes

The results achieved from these experiments agree well with Miller's results (Miller et al, 1997). For example, for a 3x3 array, Miller reported a mean fitness of 96.14 with a standard deviation of 4.52, and 50% of cases perfect. Figure 3 shows an example of a two-bit adder evolved on the 3x3 array. This is a variation on a traditional two-bit ripple-carry adder, using the same number and types of gates, but in a slightly different configuration. As discussed in (Miller and Thompson, 1998), this illustrates the ability of evolution to discover novel digital circuit designs. By using representations that allow evolution to explore a greater space of circuits than human designers would consider, we enable the generation of innovation.

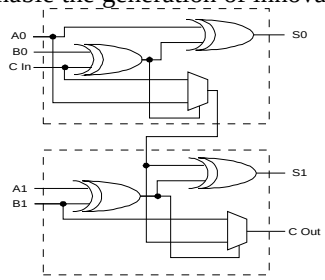


Figure 3 Example of an adder evolved on a 3x3 grid.

4 Computational Embryology

Allowing evolution greater freedom by modifying representations is just the first step towards achieving the diversity and scalability of solutions created by natural evolution. In nature, 'representations' are much more complicated: there is not a one-to-one mapping from gene to phenotypic effect. Organisms *develop* from a set of genetic instructions. The new fields of embryonics, artificial morphogenesis and *computational embryology* attempt to harness the power of such developmental processes.

Researchers have been using very simple computational embryogenies of various guises for over a decade. Current computational embryogenies can be classified into three different types: external, explicit, and implicit (Bentley & Kumar, 1999).

External embryogenies are typically, hand-designed and are defined globally and externally to genotypes. They are characterised by their fixed, non-evolvable structures. For example, Evolutionary Art systems often use external embryogenies which specify how phenotypes should be constructed using the genes in the genotypes. Similarly, Richard Dawkins' Blind Watchmaker program (Dawkins, 1987), employs a simple external embryogeny to create biomorphs, using 'eye-of-the-beholder' to provide a fitness measure, and mutation to vary the evolving shapes.

An *explicit* embryogeny specifies each step of the growth process in the form of explicit and evolvable genetic instructions. In computer science, an explicit embryogeny can be viewed as a tree containing a single growth instruction at each node. Genetic Programming (GP) uses tree structures to represent its genotypes. GP therefore, offers a simple and concise way to evolve explicit embryogenies. Typically, the genotype and the embryogeny are combined and both are allowed to evolve simultaneously. Perhaps most famous example is the work by Koza et al (1999) who used Gruau's cellular encoding for the evolution of analogue circuits. Also, Sims (1999) used an explicit embryogeny with the idea of directed graphs to specify the nervous systems (neural networks), and morphologies of virtual creatures.

An *implicit* embryogeny does not explicitly specify each step of the growth process. Instead, the growth process is implicitly specified by a set of genetic rules or instructions, similar to a 'recipe' that govern the growth of a shape. For example, de Garis (1999) describes an implicit embryogeny to evolve convex and concave shapes using a cellular automata approach. Jakobi (1996) devised an implicit embryogeny based system that employed cell division, cell movement, and diffusable proteins, in order to evolve neural net robot control architectures. Table 4 summarises the three categories of computational embrogeny.

	Outside genotypes (Non-evolvable)	**Inside Genotypes (Evolvable)**
Timing and action of every growth step is provided	EXTERNAL EMBRYOGENY	EXPLICIT EMBRYOGENY
Timing and action of growth *emerges*		IMPLICIT EMBRYOGENY

Table 4 External, explicit and implicit embrogenies.

4.1.1 Experiments

Previous work (Bentley & Kumar, 1999; Kumar and Bentley, 2000), compared the performance and scalability of different evolved computational embryogenies for the generation of tessellating tiles and letters of the alphabet. The results showed impressive scalability by an implicit embryogeny, which maintained constant genotype sizes despite evolving tessellating tiles in finer resolutions. For example, fig. 4 shows the six letters of the alphabet used as targets for evolution. Presented as 4x4, 8x8 (as shown) and 16x16 targets, they provided an assessment of the ability of evolution to produce accurate and scalable developmental processes, capable of 'growing' the desired letter.

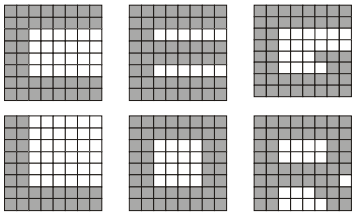


Figure 4 The pre-defined six target shapes

In these tests, two embryogenies were evolved and compared. The first was an explicit embryogeny, using GP trees to direct paths of growing cells in a phenotype grid. The second was an implicit embryogeny, which iteratively used CA-like rules to enable shapes to emerge in a phenotype grid. For full details, refer to (Kumar and Bentley, 2000). Table 5 gives the results of the experiments.

Perhaps the most significant results shown in Table 5 are the solution sizes. It is clear that the explicit embryogeny required ever-increasing tree sizes as the scale of the target shapes were increased. However, the reverse seems to be true for the implicit embryogeny, where the number of rules actually appears to decrease as the problems are scaled up. This lack of increase of solution size corroborates and confirms the results obtained in previous work which reported similar findings (Bentley & Kumar, 1999).

	4x4		8x8		16x16	
Shape	**Mean Soln. Size**	**Mean Fitness**	**Mean Soln. Size**	**Mean Fitness**	**Mean Soln. Size**	**Mean Fitness**
C	14.28 *12.70*	0.92 *1.64*	57.70 *11.47*	13.20 *12.82*	309.40 *10.00*	84.1 *53.7*
E	24.22 *11.42*	1.28 *0.32*	168.44 *11.96*	9.54 *5.89*	693.58 *6.700*	81.40 *49.40*
G	18.88 *13.86*	1.2 *0.78*	59.52 *10.98*	12.72 *12.84*	302.12 *6.000*	76.34 *52.90*
L	9.52 *11.22*	0.26 *0.58*	71.04 *9.02*	3.56 *6.38*	235.46 *8.200*	39.46 *38.40*
O	20.20 *11.60*	1.31 *0.18*	81.29 *13.29*	15.76 *9.00*	293.00 *6.400*	104.33 *48.70*
R	18.78 *12.98*	1.35 *0.60*	121.53 *12.33*	7.88 *9.92*	513.43 *6.900*	76.16 *55.80*

Table 5 Results for the target shapes. Values in *italics* denote the results for the implicit embryogeny. Solution sizes are measured in tree nodes for the explicit, and rules for the implicit embryogeny.

4.1.2 Biologically plausible implicit embryogeny

With the results of this experiment in mind, a new implicit model is now under development by the authors. The current system has been extended from the two-dimensional cellular automata, to an *isospatial grid* system. The isospatial grid, is a three dimensional coordinate system, developed by (Frazer, 1995), it uses six axis to define a point in space, yielding twelve equidistant neighbours for each point.

The system uses spheres to represent cells and builds three-dimensional morphologies by carefully placing and organising a colony of cells using a growth process. This process will use the concept of freely diffusing morphogens to allow cells to acquire positional-information. In addition, key embryological processes such as differentiation and pattern formation shall be investigated to grow morphologies. Although heavily inspired by biology, this system is not intended as a model of biological development. Instead, the work is aimed at extending the capabilities and scalability of evolutionary algorithms.

An implicit embryogeny based system is used to evolve rules that are able to grow designs in complex ways. A chromosome comprises a series of rules (genes). A rule consists of a precondition field and an action field. Each cell has a copy of the chromosome. The rules are applied (*expressed*) by matching the preconditions to a cell's state. If the preconditions are satisfied the rule is expressed. In this way, rules expressed earlier on in the development can affect other rules by switching them on or off.

Figure 5 illustrates six morphologies grown from random genomes using the new system. It should be clear that the isospatial grid enables surprisingly organic forms to emerge.

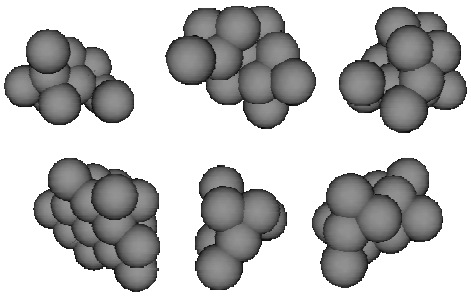


Figure 5 Six example random morphologies grown by the new system without any morphogens in the environment.

5 Artificial Immune Systems

So the use of processes that have been inspired by development in nature can increase the scalability of evolutionary algorithms. But this is not the only natural evolutionary process that researchers have recently 'stolen' from biology. Our own immune systems use evolution, enabling highly robust, adaptive and distributed detection of a vast variety of foreign pathogens. And a growing number of computer scientists have carefully studied the success of this competent natural mechanism and proposed computer immune models for solving various problems including fault diagnosis, virus detection, and mortgage fraud detection (Dasgupta, 1998).

Among these various areas, intrusion detection is a vigorous research area where the employment of an artificial immune system (AIS) has been examined (Dasgupta, 1998). The main goal of intrusion detection is to detect unauthorised use, misuse and abuse of computer systems by both system insiders and external intruders. Currently many network-based intrusion detection systems (IDS's) have been developed using diverse approaches. Nevertheless, there still remain unresolved problems to build an effective network-based IDS. As one approach of providing the solutions of these problems, previous work (Kim and Bentley, 1999a) identified a set of general requirements for a successful network-based IDS and three design goals to satisfy these requirements: being distributed, self-organising and lightweight. In addition, Kim and Bentley (1999a) introduced a number of remarkable features of human immune systems that satisfy these three design goals. It is anticipated that the adoption of these features should help the construction of an effective network-based IDS.

Previous work (Kim and Bentley, 1999a) introduced the salient functions of the human immune system with respect to network intrusion detection. In this work, we view the normal activities of monitored networks as self and their abnormal activities as non-self and design an AIS for distinguishing normal network activities from abnormal network activities.

Based on this view, we proposed a novel AIS for network intrusion detection (Kim and Bentley, 1999b), see figure 6. The AIS for network intrusion detection consists of a primary IDS and secondary IDS's. For the AIS, the primary IDS, which we view as being equivalent to the bone marrow and thymus within the human body, generates numerous detector sets. Each individual detector set describes abnormal patterns of network traffic packets and common patterns of network traffic packets when network intrusion occurs. This unique detector set is transferred to a monitored single local host. We view local hosts as secondary lymph nodes, detectors as antibodies and network intrusions as antigens. At the local hosts (secondary IDS's), detectors are background processes which monitor whether non-self network traffic patterns are observed from network traffic patterns profiled at the monitored local host. The primary IDS and each secondary IDS have communicators to allow the transfer of information between each other, see figure 6.

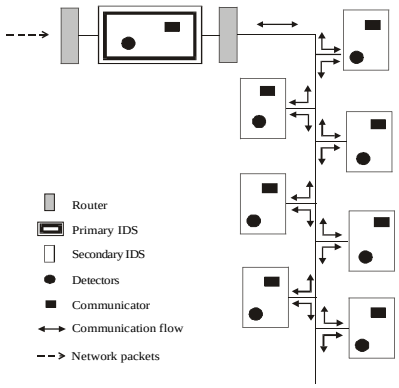


Figure 6 Architecture of the AIS for network intrusion detection.

5.1.1 Experiments

For the proposed AIS, several sophisticated mechanisms of the human immune system are embedded in three evolutionary stages: gene library evolution, negative selection and clonal selection. These processes allow the AIS to satisfy the identified goals for designing effective network-based IDS's (Kim and Bentley, 1999a). They also exploit key features of natural immune systems such as distributed detection, adaptation to new antigens and discovery of new patterns in data.

Recent work has investigated the combination of clonal selection with negative selection in the AIS. Detectors (in the form of classification rules) are evolved using the clonal selection algorithm: parent detectors compete in groups, with only the rule that matches a non-self antigen having its fitness increased. The fittest parents are randomly picked for reproduction, and if child detectors match any 'self' antigens, they are removed (negative selection), with parents generating new children. The combination of these two processes results in the evolution of a population of detectors that are clustered into niches, and that can together distinguish between 'self' and 'non-self' data. For full details, refer to (Kim and Bentley, 2001).

Three different data sets from the UCI repository[3] for machine learning algorithm benchmark work were used to test the system: Wisconsin breast cancer data (241 examples belong to 'Malignant' class and 458 examples belong to 'Benign'), the 'vote' data set (267 'democrat' and 168 'republican' examples) and the iris data set (50 examples each of 'setosa', 'virginia' and 'versicolour'). A tenfold cross-validation method was employed to prepare training sets for the AIS to evolve and test sets to evaluate detection of previously unseen non-self patterns. A detector population size of 300 was used. Each experiment was run for a maximum 50 generations.

[3] ftp:// ftp.ics.uci.edu/pub/machine-learning-databases

	TP	FP
Cancer Data	95.65 %	5.41 %
Vote Data	92.49 %	3.57 %
IRIS Setosa	99.8 %	1.2 %
IRIS Versicolor	95 %	5 %
IRIS Virginia	95.6 %	1 %

Table 6 The mean of TP and FP rates. IRIS class label in each row indicates the assigned self class.

Table 6 presents the results of the experiment, where the detector sample size was 10 and the antigen sample size was 1 (groups of ten detectors competed to detect one antigen). The detection rate of the system is described by a True Positive (TP) rate and a False Positive (FP) rate. The TP is "non-self" detection rate and the FP is the rate at which "self" is mistakenly detected by a generated detector set. The desired system will have a high TP and a low FP. The table shows the means of 10 experiments.

The results show good accuracy rates, with evolved detectors correctly identifying between 92.5 and 99.8% of non-self antigens in the test data. Equally, false-positive rates were low, with detectors mistakenly matching between 1 and 5.4% of self antigens. Further results can be found in (Kim and Bentley, 2001). Like the previous three examples, these results illustrate the benefits of exploiting the natural capabilities of evolution. The combination of immune processes produced niches of distributed detectors, which together discovered patterns in data that distinguished 'self' from 'non-self'.

6 Summary and Discussion

The four branches of evolutionary computation that we have briefly examined here form part of a new vision of EC that is now being shared by many researchers. These approaches do not force evolution to do what we think it should do by adding constraints, problem heuristics, and ill-conceived hybridisations. Instead, they all pay attention to the strengths and weaknesses of evolution. By designing systems that take full advantage of the special capabilities of evolutionary search, we are beginning to harness the power of evolution for new and more difficult problems.

The first example: creative evolutionary systems, illustrated these ideas. In order to enable evolution to generate a huge diversity of original solutions, knowledge and constraints were removed, not added. By using component-based representations, evolution is free to do what it does best: explore the search space for novelty, not find a single, global optimum. And by also allowing human interaction, evolution is able to produce good solutions to problems that cannot be encompassed in fitness functions.

The second example: evolvable hardware, also showed the creative potential of evolution in what looks set to be a very significant application for evolution in the future. By allowing evolution to consider a larger space of solutions,

it is able to find innovative new circuit designs that human designers might not think of. The results for this section included an example of such novelty: a two-bit adder evolved with a non-traditional gate configuration.

The third example illustrated some of the lessons we are continuing to learn from biology. Natural evolution is able to generate diversity and complexity in living organisms because it uses developmental processes. Like the innovations by creative evolutionary systems and evolvable hardware, evolution can generate novel developmental programs, which enable increased scalability, as was shown in the results.

Finally, the fourth example shows another way that researchers are beginning to use the natural abilities of evolution observed in biology. Our immune systems are robust, adaptive and distributed because they employ some clever evolutionary tricks. By using ideas inspired by these processes within our computers, we are able to create robust, adaptive and distributed tools, capable of discovering novel patterns that distinguish between different types of data such as normal traffic and intruders in a network.

In summary, this paper has described the evolution of "dragonflies", adders, programs of development and immune system detectors. Although diverse, these new trends in EC have something important in common: in them we are exploiting the innate abilities of evolution for exploration, innovation, adaptability and distributed search. Together, all of these new approaches will increase our abilities to harness the power of natural processes in our future technology.

Bibliography

Bentley, P. J. (2001). *Digital Biology: how nature is transforming our technology*. Hodder Headline Press, London (to appear).

Bentley, P. J. (2000). Exploring Component-Based Representations - The Secret of Creativity by Evolution? In ACDM 2000, April 26th - 28th, 2000, University of Plymouth.

Bentley, P. J. (1999a). 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.

Bentley, P. J. (Contributing Editor), (1999b). *Evolutionary Design by Computers*. Morgan Kaufman Pub. Inc., San Fran, CA.

Bentley, P. J. and Corne, D. W. (Eds) (2001) *Creative Evolutionary Systems*. Morgan Kaufmann Pub (to appear).

Bentley, P.J. & Kumar, S. (1999). Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. In *Genetic and Evolutionary Computation Conference (GECCO)* Orlando, Florida, USA.

Dasgupta, D., (1998), "An Overview of Artificial Immune Systems and Their Applications", In Dasgupta, D. (editor). *Artificial Immune Systems and Their Applications*, Berlin: Springer-Verlag, pp.3-21.

Dawkins, R. (1987). The Evolution of Evolvability. *Proceedings of Artificial Life VI*. Langton (Ed.) USA.

de Garis, H. (1999) Artificial Embryology and Cellular Differentiation. Ch. 12 in Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Pub.

Frazer, J. (1995). *An Evolutionary Architecture.* Architecture Association, London.

Furuta, H., Maeda, K. & Watanabe, W. (1995). Apllication of Genetic Algorithm to Aesthetic Design of Bridge Structures. In *Microcomputers in Civil Engineering v10:6.* Blackwell Publishers, MA, USA, 415-421.

Hollingworth, G., Smith, S., and Tyrell, A. (2000) The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic. In *Proceedings of the Third Int. Conf. on Evolvable Systems*, Springer, pp. 72-79.

Jakobi, N. (1996) Harnessing Morphogenesis. University of Sussex, Cognitive Science Research Report #429, Brighton, UK.

Kim, J. and Bentley, P., (1999a), "The Human Immune System and Network Intrusion Detection", *7th European Conference on Intelligent Techniques and Soft Computing (EUFIT '99), Aachen, Germany*.

Kim, J. and Bentley, P., (1999b), "The Artificial Immune Model for Network Intrusion Detection, *7th European Conference on Intelligent Techniques and Soft Computing* (*EUFIT'99), Aachen, Germany*.

Kim, J. and Bentley, P. J. (2001), The Artificial Immune System for Network Intrusion Detection: An Investigation of Clonal Selection with a Negative Selection Operator. Submitted to *CEC2001, the Congress on Evolutionary Computation.*

Koza, J.R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. (1999). *Genetic Programming III*. San Francisco, CA: Morgan Kaufmann.

Kumar and Bentley (2000). Computational Embryology: Past, Present and Future. To be published as an invited chapter in Ghosh and Tsutsui (Eds) Theory and Application of Evolutionary Computation: Recent Trends. Springer Verlag (UK).

Levi, D. (2000) "HereBoy: a fast evolutionary algorithm." In *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, IEEE, pp. 17-24.

Miller, J. F., Thompson, P., and Fogarty, T. (1997) Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study. In *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*, Wiley, Chapter 6.

Miller, J. F., Thompson, P. (1998) Discovering Novel Digital Circuits using Evolutionary Techniques. In *IEE Colloquium on Evolvable Systems*, IEE.

Rowbottom, A. (1999). Evolutionary Art and Form. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Publishers Inc., San Francisco, CA.

Sims, K. (1999). Evolving three-dimensional Morphology and Behaviour. Ch. 13 in Bentley, P. J. (Ed.) *Evolutionary Design by Computers*. Morgan Kaufman Pub.

Thompson, A. (1996) Silicon Evolution. In *Proceedings of Genetic Programming 1996*, MIT Press, pp. 444-452.