"Going Back to our Roots": Second Generation Biocomputing

JON TIMMIS[1⋆], MARTYN AMOS[2†], WOLFGANG BANZHAF[3‡], ANDY TYRRELL[4¶]

[1] *Department of Electronics and Department of Computer Science, University of York, YO10 5DD, UK*
[2] *School of Engineering, Computer Science and Mathematics, University of Exeter, Exeter EX4 4QF, UK*
[3] *Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, A1B 3X5, Canada.*
[4] *Department of Electronics, University of York, YO10 5DD, UK*

Received 16th December;

Researchers in the field of biocomputing have, for many years, successfully used the natural world as inspiration for developing systems that are robust, adaptable and capable of generating novel and even "creative" solutions to human-defined problems. However, in this position paper we argue that the time has now come for a reassessment of how we exploit biology to generate new computational systems. Previous solutions (the "first generation" of biocomputing techniques), whilst reasonably effective, are crude analogues of actual biological systems. We believe that a new, inherently inter-disciplinary approach is needed for the development of the emerging "second generation" of bio-inspired methods. This new *modus operandi* will require much closer interaction between the engineering and life sciences communities, as well as a bidirectional flow of concepts, applications and expertise. We support our argument by examining, in

⋆ email: jtimmis@cs.york.ac.uk
† email: M.R.Amos@exeter.ac.uk
‡ email: banzhaf@cs.mun.ca
¶ email: amt@ohm.york.ac.uk

this new light, three existing areas of biocomputing (genetic programming, artificial immune systems and evolvable hardware), as well as an emerging area (natural genetic engineering) which may provide useful pointers as to the way forward.

Key words: bio-inspired computing, genetic programming, artificial immune systems, evolvable hardware, natural genetic engineering, biological plausibility

1 INTRODUCTION

> Natural organisms are, as a rule, much more complicated and subtle, and therefore much less well understood in detail, than are artificial automata. Nevertheless, some regularities which we observe in the organisation of the former may be quite instructive in our thinking and planning of the latter – John von Neumann, 1948 [124].

Even just after the 2nd world war, scientists were already thinking about the conceptual cross-over between natural and artificial systems. Von Neumann and Turing were but two of the pioneers who contributed to the emergence of *bio-computing* – the extraction of computational principles or methods from natural, biological systems.

Biologically-inspired computational methods such as artificial neural networks and genetic algorithms have been successfully used to solve a wide range of problems in the engineering domain. We can think of such methods as comprising the "first generation" of biocomputing techniques, in that they rely on (often very) crude approximations of the fundamental underlying biological principles (for example, the basic crossover operator used in genetic algorithms).

Such crude models have, up until now, been accepted for several reasons: the first is that they have produced solutions that are considered "good enough" in terms of their fitness for purpose. The second reason is borne out of necessity, in that a sufficiently-detailed description or understanding of the underlying biological system has, so far, eluded us. Both reasons for acceptance of the *status quo* are now, we believe, beginning to be eroded, both by a growing unease at the limitations of current nature-inspired models, and by the speed at which our understanding of biology is increasing. We believe that the time is right for the development of a "second generation"

of bio-computational methods that draw much more closely on the growing understanding of their biological inspiration.

One possible explanation for the lack of recent progress in nature-inspired computing may be that the respective disciplines parted company far too prematurely, As the report of a 2001 EPSRC workshop minuted, "One of the major challenges facing collaborative work in biologically inspired computing systems is the temptation to diverge far too early. This has been the case to some extent in genetic algorithms where this method has not moved forward as hoped because of premature divergence of the computing and biology communities"[1].

If the second generation of biocomputing is to emerge, it is perhaps the case that a new convergence of disciplines is required, to re-ignite the initial spark of interest that passed between them. Developments in systems (and now *synthetic*) biology are driving advances in biology, engineering and computer science – crucially, these breakthroughs are no longer unidirectional, in that expertise and concepts flow in a one-way stream from one discipline to another. The new *systems-level* philosophy that is beginning to dominate 21st Century science dictates that artificial boundaries between disciplines must be transcended or even demolished. A first step in this process might be to revisit our perspective on nature-inspired computing, and perhaps even reinvent it from the bottom-up.

The UK research community has proposed a number of Grand Challenges for Computer Science research and ambitious plans for the development of a variety of research areas. Grand Challenge 7 (GC-7) [107] addresses the area of Non-Classical Computation, which encompasses both biologically inspired paradigms and the it direct exploitation of the natural world for computation (for example, DNA, cellular and quantum computing). Part of its ambition is to – once again – bring together disciplines that have prematurely drifted apart. It has become clear that not all bio-inspired approaches are the same, each having their own contribution to make to the scientific community.

In this position paper we highlight two main themes, the first of which being the importance of *the precise nature of the particular biological system that we are trying exploit* – different systems contain different mechanisms and functionalities that give rise to different properties. The second – and main – argument in this position paper is that *high level abstractions of underlying biology are no longer sufficient*, and that a *deeper* level of understanding is required. It will become necessary to first *understand* and then *exploit* the full underlying richness of biological systems, through an interdisciplinary approach that combines biology with computation and engineering

in a *synergistic* manner.

In order to expand on these themes, this paper will first discuss two exemplar bio-inspired systems (section 2); specifically Genetic Programming (GP) – section 2.1 – and Artificial Immune Systems (AIS) – section 2.3. We reflect on each of these themes by describing how they have evolved as computational paradigms over the years, and how they have addressed (or failed to address) the main arguments given above. We also emphasise the importance of considering the *deployment* of artificial systems; for example, software and hardware systems have very different practical implementation considerations and constraints. We discuss these particular issues in section 3 – once again reflecting on the arguments outlined above, but this time in the context of evolvable hardware (EHW). Finally, in order to provide an over-arching perspective, we examine the area of Natural Genetic Engineering (section 4 and 5), a biologically inspired approach to studying development and evolution that is *fundamentally* rooted in biology, and which offers feedback of real benefit to both computer scientists *and* biologists, as opposed to the often one-way flow of information and insight "absorbed" by practitioners working on the other methods under discussion.

2 CAPTURING EVOLUTION AND IMMUNITY

2.1 Genetic Programming

Genetic Programming is a technique for "breeding" computer programs, loosely based on Darwin's theory of natural selection [68]. More precisely, it is an algorithmic approach based on a new synthesis of evolution, which includes the transferral of genetic traits from one generation to another by way of molecular mechanisms based on DNA (the *genotype*) and its subsequent transformation into *phenotypes* [9]. This approach naturally follows other successful applications of the evolutionary paradigm to the solution of optimization problems; namely Genetic Algorithms [53], Evolutionary Programming [32] and Evolutionary Strategies [96]. These earlier applications of the same fundamental idea differ from GP in two important ways; (1) the basic optimisation goal (primarily, the discovery of a single, unchanging optimum, as opposed to a potential "moving target"), and (2) the nature of the underlying computational substrate (fixed-length, fixed-representation data structures, as opposed to a rather more dynamic GP structure).

GP is therefore different to previous methods in that the complexity of both the *task* (be it optimising the behaviour of a robot or an algorithm) and that of the *data structure* (a metric perhaps expressed in terms of the length

of code) are *variable* over the course of the evolutionary algorithm. Both complexity considerations are tightly coupled: if the complexity of a *task* changes over time (or, indeed, is unknown or unknowable), then it must be possible to accordingly modify the complexity of the *solution representation*.

We first consider the issue of solution *representation*. Initial applications of GP typically employed the so-called *parse-tree* representation, one of the clearest (and most basic) examples of an evolvable data structure. Later, a number of other data structures were proposed, these being perhaps even more amenable to artificial evolution. Prominent among these was a linear sequence of instructions from an imperative programming language [10, 87], and a generic graph structure (a special case of which being the parse-tree) [114]. After an initial flurry of publications on methods, during which different operators and selection methods were examined in addition to simply the data structure, the field of GP has "settled down" and recently turned to application and exploitation of the basic method. Some impressive early applications paved the way for future inquiries [43, 103, 44, 70] and recently the field has matured to such a degree that applications rivalling and even superceding the performance of human designers are routinely demonstrated. For example, the prominent researcher John Koza has applied for patent protection of a particular design generated by his highly parallel computational approach toward circuit design. Others have succeeded in designing antennae for use in space missions that out-perform human-generated designs [77]. The GECCO conference [2] now features an annual competition dedicated to human-competitive or human-exceeding applications of GP and other evolutionary methods.

Although the field of GP is now well-established, its maturity is still open to question. Since GP is an abstraction of a "snapshot" of biology, it follows that the field should continue to develop in the light of new findings in the domain of the life sciences. As such, the field of GP has not yet reached a steady state, but should be continually driven by further progress in our understanding of biology. We now consider in more detail how this might occur.

2.2 Reflections on Genetic Programming

Implications of Exploiting Evolution

One of the main strengths of GP lies in the *adaptability* of the "genetic" representation of solutions. However, such flexibility is limited mainly to changes in the *size* of a proposed solution in terms of the number of "elements" used (e.g., nodes in a graph or tree, lines of code in a linear representation). This

facility gives rise to a phenomenon whereby the average length of evolved programs (i.e. solutions) tends to continually grow over time unless artificially restricted [6], and that the resulting code does not always manifest itself in additional algorithmic complexity. This growth of evolved data structures used to represent algorithmic solutions is attributed to so-called *neutral code* (referred to as *introns*, after non-coding DNA found in real gene sequences) [4, 72].

Subsequent investigations established that this *emergent* property of GP [9]

- was ubiquitous, being found in multiple GP implementations
- had both positive and negative effects on algorithm behavior
- was implicated in inefficient use of computer resources (i.e., processor time, memory, disc space)
- needed an explanation based on the particular GP method used

A consensus is beginning to emerge as to the reasons for the emergence in GP of the intron phenomenon – that extra code helps to protect individuals against potentially harmful crossovers or mutations, by reducing the probability of a crucial component being disrupted by one of these operations. Remedies have been proposed to prevent its negative effects (e.g., homologous crossover [47]), and other methods have been used to deliberately induce the effect (e.g., explicitly defined introns [88]) in order to further study its implications. Following the tradition of other evolutionary computation approaches, the basic question of "building blocks" and their growth dynamics has been examined in the context of GP. In order for such analysis to be successful, the question of how length-varying evolutionary algorithms work needs to be addressed at a fundamental level, and significant and substantial steps towards this have already been made [90, 91]

Code-growth (often referred to as "bloat") is not the *only* emergent phenomenon observed in GP, but it is perhaps the most obvious. Closer inspection of solutions bred by GP approaches has found that evolved code shows internal patterning reminiscent of the repetitive structure of natural genomes. Observed first in linear sequences of code [73], this spontaneous structure formation has been now confirmed in other representations [8].

In recent years, exciting new developments have been reported by virtue of merging approaches from GP with those from the Artificial Life community [106]. The application sphere of GP (and other evolutionary approaches) is

expanding every year, and the amount of work published is growing in turn. Yet there is also an "underground" stream of work that is not published or patented, and is instead held as protected trade secrets. This body of research generally refers to applications in the financial sector, where a slight "edge" over the competiton (e.g., in terms of evolved predictors for the stock market) may yield a significant financial return. The number of solid GP applications now stands in the hundreds, ranging from elementary particle physics [75] to bankruptcy prediction [79]. However, there still exist a number of domains in which further progress will be needed before GP can make a significant impact. First and foremost, present GP techniques do not scale well. Useful applications are currently restricted to short programs of a complexity comparable to 50 to 250 "standard" lines of code. If GP methods are to have a long-term future, then future work must focus on scalability in order for them to move into new domains of applicability. We consider how this might be achieved in the next section.

Levels of Abstraction Employed - Getting Back to the Biology

In most competitive programs that have been evolved to date, some sort of structuring of the evolutionary process has turned out to be necessary. It can been argued that, without explicit inclusion of a developmental process that might faciliate the emergence of genomic structuring and the "upscaling" of solutions through growth, GP will continue to suffer from the twin problems of scalability and the lack of potential for modularity of code [12, 15, 55]. The inclusion of such a process, however, does not come for free. Such major changes to (and potential improvement of) the basic GP method will necessitate a significant complication of the genotype-to-phenotype mapping process – of which current practitioners are understandably wary. We believe, however, that without consideration of fundamental developmental processes in evolution [11], further progress in GP will become stalled.

As advances in molecular biology have made clear, the genomes of various organisms contain multiple levels of complexity. Although only 1-3% of the human genome is *translated* into protein and thus "codes" for something, more than 50% of the genome is at least transcribed (i.e., "read", in preparation for conversion into a protein) [62]. This confirms the suspicion that we as scientists have barely started to understand the formation of phenotypes from genetic information [129]. One fundamental question that might be asked is this: Is transcription a more important step in the formation of the phenotype than translation into proteins? In addition, phenotypic differences

can be present in multiple individuals with identical genetic makeups [133]. This points to the possible additional consideration of *epigenetics* as being of primary importance if we want to understand evolution [130], and thus incorporate this extra power into GP. Evolution, however, is not the only adaptive mechanism available to us for study, abstraction and application. The neural system, the endocrine system and the immune system are also some of many systems deserving of attention. In the next section, we turn our attention to the last of these.

2.3 Artificial Immune Systems

The immune system is a complex mechanism that undertakes a multitudeof tasks, and its abilities have inspired computer scientists to build artificial systems that mimic its properties [26]. This field of research – Artificial Immune Systems – has seen the application of immune-inspired algorithms to problems as diverse as as robotic control [71], network intrusion detection [33, 65], fault tolerance [22, 7], bioinformatics [85] and machine learning [67, 128]. From a computational point of view, the immune system has many desirable abilities that could usefully be endowed on artificial systems. These properties include: robustness, adaptability, diversity, scalability, and multiple interactions on a variety of timescales. We now consider these properties in more detail, and see how they might be usefully utilised.

A Brief History of Artificial Immune Systems

The origins of AIS has its roots in the early theoretical work of Farmer, Perelson and Varela [30, 89, 123], in which a number of network models were proposed to describe the maintenance of immune memory in the absence of antigens. These models, whilst controversial from an immunological perspective, attracted much interest from the computer science community. Two early contributors to the fusion of immunology and computing were Hugues Bersini and Stephanie Forrest. Some of the early work by Bersini [16, 17] is very well-rooted in immunology, and this is also true of the initial contributions of Forrest [34, 50]. All of this work formed an excellent foundation for future work in the area of AIS. Bersini, concentrated on immune network theory, examining how the immune system maintained its memory and how one might build models and algorithms mimicking that property. Forrest's work focussed mainly on computer security (in particular network intrusion detection) [33, 52], and formed the basis for a large body of subsequent work.

In the mid-1990s, researchers in the UK began to investigate the nature of *learning* in the immune system, and how this might be used to create machine

learning algorithms [24]. Initial results were very encouraging, and this success was further developed by the application of immune system ideas to the classification of DNA sequences [56] and the detection of potentially fraudulent mortgage applications [57]. Immune network-based machine learning gathered further momentum, notably in [117, 121] where the Hunt and Cook system just mentioned was totally rewritten, simplified and applied to unsupervised learning (in a manner very similar to cluster analysis). This thread of work on machine learning generated more work in the unsupervised domain, but this time focussing mainly on dynamic clustering [132, 83]. At the same time, other immune-inspired work [49] used associative memory tecniques to track "moving targets" in databases. In the supervised learning domain, very little happened until Watkins [125] developed an immune based classifier known as AIRS. This important system (later augmented in [128]) has evolved into a parallel and distributed learning system [126], and is one of the real "success stories" of immune inspired learning [41, 40, 127].

For a full review of AIS and its applications in the past five years, the reader is directed to various reviews in the literature [25, 119, 26, 37, 48]. The International Conference on Artificial Immune Systems (ICARIS) conference series began in 2002, and is now well-established [118, 122, 86, 60]. This series is probably the best source of reference material, in terms of both the variety of AIS applications and theoretical developments in the field.

2.4 Reflections on Artificial Immune Systems

Implications of Exploiting Immunology

The work of Forrest et al. [35] led to a great deal of research into immune-inspired *anomaly detection* systems for the purposes of detecting computer viruses[33]. These results hinted at the possibility that the immune approach would indeed be beneficial, as they showed that both known and novel virus intrusions could be detected. However, given the typical representation used (binary), and that the *r-contiguous bits* matching rule was typically used to compare contiguous regions binary strings, there arose issues of computational efficiency. The subsequent *r-chunk* rule allowed efficient generation of a set of detectors for the non-self space (in Hamming shape space), and more efficient methods were also developed in *real*-valued shape space [39]

Unlike in GP (section 2.2), where code bloat was a problem, in the case of this particular application of AIS, the main problem became that of generating enough detectors to effectively covering the search space; In essence, there arose an exponential relationship between the size of the *self* data and the number of detectors that could be generated. Work in [110, 111] presented

an in-depth theoretical analysis of the *negative selection* algorithm over real and Hamming shape spaces. The authors suggest that, over the Hamming shape-space, the approach is not well suited for real-world anomaly detection problems. Problems seem to arise when the generated detector set *under-fits* the training data exponentially for small values of r (where r is the size of the chunk). It has been suggested that, in order avoid this under-fitting behavior, the matching threshold value r must lie near l (the length of the string). However, the consequence of this is that the detector generation process once again becomes practically infeasible, since all proposed detector generating algorithms have a runtime complexity which is exponential in r. In addition to these theoretical arguments, simple experimental comparisons have been made between the various negative selection approaches and a one-class support vector machine (SVM). When assessing the work of [39] (the real-valued negative selection algorithm with variable-sized detectors), experiments revealed not only that the classification performance of the method crucially depended on the size of the variable region, but that the one-class SVM provides *as good*, if not better results. As argued in [48], it is not clear that the application of AIS to network security has, so far, yielded any significant breakthroughs.

Another example application of AIS is drawn from the work of Timmis *et al.* [120]. In [120], an immune network inspired algorithm was proposed that was capable of performing unsupervised learning. Initial results were very encouraging, but further investigations in [66] highlighted a number of problems, and identified a behavioral pattern different to that observed in the original work. Subsequent investigations discovered that the algorithm would naturally discover the strongest pattern within the application data set, and that the network would eventually converge to a single cluster (an undesirable property). Pressure within the network was too great for "weaker" cells to survive, which arose because of a naive implementation of network interactions, where stimulation and suppression occurring between cells was not effectively balanced. As highlighted in [38], a too simplistic approach had been taken to the representation of data vectors, and, in particular, to the definition of their interactions.

Levels of Abstraction Employed - Getting Back to the Biology

The original AIS were developed with an interdisciplinary slant, taking care to generate algorithms that were '"faithful" to their immunological roots. For example, Bersini [16, 17, 18] pays clear attention to the development of im-

mune network models, and then applies these models to a control problem characterised by a discrete state vector in a state space. Bersini's proposal relaxes the conventional control strategies, which attempt to drive the process under control to a specific *zone* of the state space; instead, he argues that the metadynamics of the immune network is akin to a meta-control whose aim is to keep the concentration of the antibodies within a certain range of viability, so as to continuously preserve the *identity* of the system.

There exist many other examples of similar interdisciplinary work, such as the development of immune gene libraries and a subsequent *bone marrow* algorithm employed in AIS [50], the development of the negative selection algorithm and its first application to computer security [35]. However, in more recent years, work on AIS has slowly drifted away from more biologically-appealing models and attention to biological detail, with the focus shifting to a more engineering-oriented approach. This has led to systems that are examples of "reasoning by metaphor" [108]. These include simple models of clonal selection, immune networks and negative selection algorithms, as outlined above. For example, the clonal selection algorithm (CLONALG) [23], whilst intuitively appealing, lacks any notion of interaction of B-cells with T-cells, MHC or cytokines. In addition, the large number of parameters associated with the algorithm, whilst well understood, make the algorithm less appealing from a computational perspective. aiNET, again, whilst somewhat affective, does not employ immune network theory to any great extent. Only suppression between B-cells is employed, whereas in immune network theory, there exists suppression and stimulation between cells. With regard to negative selection, the simple random search strategy employed – combined with a simple binary representation – makes the algorithm so computationally expensive that it is almost unusable in a "real world" setting [112].

However, recent work by the Danger Team [3] has started to address this imbalance between theory and biological reality. For example, [42] reports initial explorations into the use of *dendritic* cells (a type of cell found in the innate immune system) as a mechanism for identifying dangerous (or anomalous) event in a data stream. While this research is still preliminary, and currently works only on static data, it offers a great deal promise, and may go some way towards offering the possibility of real breakthroughs in the intrusion detection subfield of AIS research. In related work, Bently [14] proposes an *artificial tissue*; a form of representation of the data space that can evolve and adapt over time. Again, this is very preliminary work, but could provide a useful "bridge" between data and the immune algorithm itself. In addition to this, work in [109] proposes a "conceptual framework" for the development of

AIS (although it could be generalised to any bioinspired approach). This proposes greater interaction between computer scientists, engineers, biologists and mathematicians, in order to gain better insights into both the workings of the immune system, and the applicability (or otherwise) of the AIS paradigm. These interactions should be rooted in a sound methodology in order to fully exploit the synergy.

Whilst the immune system is clearly an interesting system to investigate, if viewed in isolation many key emergent properties arising from interactions with *other* systems will be missed. Such systems do not operate independently in biology; consideration should therefore be given to the interactions of the immune, neural and endocrine systems, and how – together – they facilitate emergent properties [104, 20, 102]. Immune, neural and endocrine cells express receptors for each other, and this allows interaction and communication between cells and molecules in "partner" systems. It appears that products from the immune and neural systems can exist in lymphoid, endocrine and neural tissue *at the same time*. This indicates that there is a bi-directional link between the nervous system and immune system, and, therefore, it would seem that both endocrine and neural systems can affect the immune system. There is evidence to suggest that, by stimulating areas of the brain, it is possible to affect certain immune responses, and also that stress (which is regulated by the endocrine system) can suppress immune responses: this is also reciprocal, in that immune cells can affect endocrine and neural systems. The action of various endocrine products on the neural system is accepted to be an important stimulus of a wide variety of behaviours. These range from behaviours such as flight and sexual activity, to sleeping and eating [84].

From a computational perspective, then, how may these finding inform the AIS community? It should be possible to explore the role of interaction between the three systems just mentioned – one potentially interesting avenue would be to design an AIS to help select the types of components which will be most useful when *added* to a control system at any moment ("differentiation") and to *remove* components when they are proving harmful to the control system ("apoptosis", or cell death). The *biological* immune system cells select the particular action to perform by detecting properties of the cells and chemical environment, through molecular interactions at membrane receptors. In an *artificial* system, similar properties could be detected by looking at activation states of artificial neurons and endocrine cells, as well as global state information such as current consumption and battery levels. Thus, the artificial immune system components could have the ability to make similar

decisions to those of the real biological system.

3 WHAT ABOUT THE MEDIUM?

We have now reviewed two bioinspired paradigms, one well established (GP - section 2.1) and one relative newcomer, AIS - section 2.3). So far, there has been no consideration of the *physical* nature of the system on which these models are developed. In this section, we address this by reviewing the area of Evolvable Hardware, observe how evolution may be used in hardware-based systems, and describe the specific considerations encountered when developing such systems.

EHW is a method for electronic circuit design that uses inspiration from biology to *evolve* rather than design hardware. At first sight this would therefore seem very appealing. As we have already seen, natural systems tend to be robust, adaptable, complex and reliable. Human designs tend only to beat natural systems in terms of their optimality. Evolvable hardware is a new field that brings together reconfigurable hardware, artificial intelligence, fault tolerance and autonomous systems. It refers to hardware that can change its architecture and behaviour dynamically and autonomously through interaction with its environment. Ideally, this process of interactions and changes should be continuous and open-ended. Hardware is therefore created through an evolutionary process of continual refinement, which only terminates when a sufficiently "good" individual has been found.

Both evolvable and *evolved* hardware make use of evolutionary techniques in order to produce a system that performs to some specification (here, we mainly consider the latter); both are, to some extent, autonomous, and both may have properties that endow the final system with *limited* fault tolerance. The major (but not the only) difference between the two is that *evolved* systems do not change after a "good" individual has been found, and are therefore rather static in terms of their design (as are most human designs). *Evolvable* systems possess the capability to change throughout their lifetime. Hence, as the system itself changes (e.g., components fail), or as the environment changes (e.g., the temperature changes), an evolvable system is able to adapt to these changes and continue to operate effectively.

3.1 A Brief History of Evolutionary Hardware Systems

It may argued that artificially intelligent system design should address features such as autonomy, adaptability, robustness, and fault-tolerance. Autonomous robot navigation in dynamic environments represents a very chal-

lenging task, and needs to take into account such factors. Conventional approaches based on *off-line* learned control policies do not generally work appropriately when implemented in real time environments. For example, the actual hardware system implementing the evolved behaviour may well not accurately match the simulation environment used during evolution. The sensors and actuators used in the real system may have different characteristics to those used in the simulation (e.g., infra-red sensors in a bright environment would operate differently to those in a dark or changing light environment). The development of EHW – the application of evolutionary algorithms [31] to automatic design and or reconfiguration of electronics circuits [134], presents a promising approach to the problem of adaptation in unknown/changing environments.

Two methodologies have been established for the design of EHW: *Extrinsic* and *intrinsic* [116, 81, 74, 46, 54]. In the former case, both the evolutionary process as well as the fitness evaluation of each individual (the circuit) is simulated in *software*. The entire design is undertaken off-line, and once the evolutionary process has completed, the "best" member of the final population is downloaded onto the hardware. In the latter case, the evolutionary process maybe executed in software, but each individual is executed and evaluated in *hardware* [46, 54]. Developments in electronic devices such as the Field Programmable Gate Array (FPGA) – reconfigurable devices with no pre-determined function [101] – have enabled theoretical ideas of intrinsic evolution to become a reality in the last few years. Each individual is represented as a bit string (genotype) that is downloaded to the chip as configuration data. This data includes a definition of each cell's functionality as well as the topology of the system.

Higuchi's group in Japan have taken a different approach to EHW. Rather than use extrinsic evolution, or intrinsic evolution on Commercial, Off-The-Shelf (COTS) chips, they have developed a single Large-Scale Integrated (LSI) chip that is specifically designed to support evolvability [59]. They have developed a gate-level chip that consists of genetic algorithm hardware, reconfigurable hardware logic and the control logic required. This chip, and variants of it, have been applied to a number of applications, including and artificial hand controller, autonomous robotics, data compression of image data, analogue chip design for cellular phones, optical system adjustment and adjustment of clock timings [51, 63, 76]. What these results show is an increased cycle time in the evolutionary process, due to the specialised optimisation that has taken place in the hardware.

Evolutionary computation – using a binary representation – appears to be

convenient when applied to EHW, since it's substrate perfectly matches the nature of the configuration bits used in FPGAs. There are, however, huge problems associated with the evolution of large circuits (or what today are probably considered *small* circuit designs) due to problems of *scaling*. That is, with direct genotype-phenotype mappings such as these, as the circuit complexity increases so does the size of the genotype and the size of the search space (as we have already seen). A number of papers have been published to evolve on-line FPGA-based robot controllers using these methods [115, 64, 45, 113] using COTS. One of the main problems of evolving on a FPGA is the genotype-phenotype mapping. Effective methods to solve this problem using intrinsic EHW have been proposed. In intrinsic EHW, the fitness is evaluated on target hardware. Therefore, changes in environment are reflected *immediately* in the fitness evaluation. For example, the problem of adaptation of autonomous robot navigation in changing environments consists of finding a suitable function F (the controller) which maps the inputs from sensors to the outputs (control signal to the motors). Another evolutionary approach to EHW is to use GP – techniques such as Cartesian Genetic Programming (CGP) [80] and Enzyme Genetic Programming (EGP) [78] have been applied extensively to EHW. A recent development, and an attempt to move back to biology is considered in [21].

One criticism of CGP (and GP in general) is that the location of genes within the chromosome has a direct or indirect influence on the resulting phenotype. In other words, the order in which specific information regarding the definition of the GP is stored has a direct or indirect effect on the operation, performance and characteristics of the resulting program. Such effects are considered undesirable, as they may mask or modify the role of the specific genes in the generation of the phenotype (or resulting program). Consequently, GPs are often referred to as possessing a direct or indirect *context representation*.

An alternative representation for GP, in which genes do not express positional dependence, has been proposed by Lones and Tyrrell [78]. Termed *implicit context representation*, the order in which genes are used to describe the phenotype (or resulting program) is determined *after* their self-organised binding, based on their own characteristics and not their specific location within the genotype – a much more biologically-realistic mechanism. The result is an implicit context representation version of traditional parse-tree based GP, termed *Enzyme Genetic Programming*. The authors have since implemented an implicit context representation of CGP, termed Implicit Context Representation Cartesian Genetic Programming (IRCGP), specifically for the

evolution of image processing filters [21].

In many evolutionary algorithms, whether intrinsic or extrinsic, once the final criteria have been met (the required fitness level or the maximum number of generations) the evolutionary process stops, and the best of population is used in the implementation. An alternative approach is to allow continuous evolution throughout the lifetime of a system. Once a member has been chosen for implementation, the evolutionary process does not stop. Such a continuous process allows a system to be more responsive to environmental changes. For example, evolution can cope with errors *during runtime*. The system fitness might temporarily drop at the instant the error is activated, but the evolutionary process autonomously deals with this drop in fitness and recovers back to an acceptable level (and thus an acceptable level of functionality) over a number of generations.

3.2 Reflections on EHW

Implications of Exploiting Evolution

Original design of digital circuitry is an area in which the EHW community has experienced limited success. To fully appreciate why, it is important to understand how industry performs digital design. The overwhelming majority of digital design today is carried out using electronic design automation (EDA) tools. Complicated designs are usually implemented in an FPGA, or in an application specific integrated circuit, where the device density permits high-speeds in small packages. The complexity of these designs makes hand design methods impossible. EDA tools automate the design process, thereby reducing design time, improving yields, and reducing nonrecurring costs.

EHW practitioners need to understand that the FPGA design flow is in place and widely used throughout the integrated circuit industry today. In essence, this means the EHW community needs to demonstrate substantial and significant advantages over an established method before making any real inroads, and this necessity presents the main challenge to any EHW method for circuit synthesis. An additional – but fundamental – issue all EHW users have to face is *scalability*. A typical ’“big” EHW system might be a few 100 transistors (1970’s technology for the microprocessor companies). Current chip designs contain on the order of 100,000,000 transistors.

Levels of Abstraction Employed - Getting Back to the Biology

The area of adaptive system design and control is where EHW methods have the greatest potential for digital, analog and mixed signal systems. Circuitry

can be adapted, i.e., reconfigured, to take on new roles, or for fault recovery. Consequently, the focus should be on *adaptation* for fault recovery operations.

Circuits are adapted *in-situ*. What makes this environment particularly problematic is the fact that the user almost never has complete knowledge about why the original circuitry failed. Obviously, faults can degrade a circuit's performance, but so can any change in the operational environment. Regardless of the cause of a fault, reconfiguration done *in-situ* is especially challenging for two reasons: (1) faults can be hard to detect and isolate, and (2) the reconfiguration function may itself be compromised by the fault. However biological systems seem to operate, for the majority of their time, continually faced by all of these issues (and more besides).

At the present time, a significant portion of EHW-based fault recovery investigations rely on simulations and usually fairly simplistic models of biological evolution. As with many evolutionary-type systems, genotype to phenotype mappings tend to be rather simplified, usually one-to-one. As mentioned previously, developmental processes are seldom considered (although, as we have already pointed out, this can lead to its own issues). Investigations are needed to develop intrinsic evolutionary methods for autonomous systems with limited resources. This should also include recovery techniques. Like biological systems, these evolutionary methods should not stop when one good solution is found. We need to consider how we can incorporate biological *open-ended* evolution into our systems.

More work needs to be done on developing EAs that can intrinsically evolve circuit configurations with *imprecisely*-defined performance objectives (imprecise in the sense that one does not know in advance what level of performance can be achieved). Evolving benign configurations, where further damage is contained and controlled, is also of interest. Again, more accurate models of the equivalent biological processes need to be considered for the eventual benefit of our hardware systems.

Studies are needed to determine how effective EHW-based recovery methods are when the computing resources they run on are degraded by environmental conditions. Can we somehow use homeostasis-type ideas to make our systems adapt intelligently – to ensure critical functions are kept operational – at the expense of other, less critical, functions - for example, the immune-endocrine-neural system (section 2.4)? Many fault recovery scenarios involve injecting *arbitrary* faults into an operating circuit, e.g., a randomly chosen switch in a circuit is forced open. It is not clear if such a fault is likely to occur in isolation or whether it results from some other fault. This ambiguity

leads to the development of recovery methods that may have limited usefulness. A major issue with all fault-tolerant systems is error detection. This is mostly ignored in EHW systems. Artificial Immune Systems should have a role to play here and more should be made of the combination of *multiple* bio-inspired ideas, as we have already discussed in section 2.4

4 AT THE INTERFACE OF BIOLOGY AND COMPUTING

Biological systems such as individual cells are capable of performing amazingly intricate and sophisticated information processing tasks, including pattern recognition and distributed memory storage (previously described in the context of the immune system), pattern *formation* [19], distributed communication [13] and adaptive control [105].

As we have already seen, descriptions of cellular systems may be usefully abstracted and applied to the solution of human-defined computational problems. In particular, studies of bacterial attraction and movement have been successfully applied to (among other problems) the training of artificial neural networks [27] and the design of aircraft aerofoils [82]. In addition, actual living cells have also been directly *engineered* to perform simple computational tasks. In 1999, Weiss *et al*. [131] described a technique for mapping digital logic circuits onto genetic regulatory networks such that the resulting chemical activity within the cell corresponded to the computations specified by the desired digital circuit. There was a burst of activity in 2000, when two papers appeared in the same issue of *Nature*, both being seminal contributions to the field. In [29], Elowitz and Leibler described the construction of an oscillator network that caused a culture of *E.coli* to periodically glow by expressing a fluorescent protein. Crucially, the period of oscillation was slower than the cell division cycle, indicating that the state of the oscillator was transmitted from generation to generation. In [36], Gardner *et al.* implemented a genetic toggle switch in *E.coli*, the switch being flipped from one stable state to another by either chemical or heat induction. These "single cell" experiments demonstrated the feasibility of implementing artificial logical operations using genetic modification. In [97], Savageau addresses the issue of finding general design principles among microbial genetic circuits, citing several examples. Several more examples of successful work on cellular computing may be found in [5].

It is clear, therefore, that working at the interface of cellular biology and engineering/computer science can generate tangible benefits in terms of "real world" applications. However, these applications do not bring us any closer to

a real break-through in terms of a completely novel computational paradigm. Moreover, with very few exceptions, it is rarely the case that such studies add anything to our overall understanding of the underlying biological system. Here, we argue that the development of novel biological algorithms should be, at least in part, motivated by a desire for longer-term insights, and not just short-term applications. Given the work presented so far in sections 2.1 and 2.3, there seems to be clear evidence that no matter how appealing creating applications can be, there are some inherent difficulties in adopting biologically inspired approaches, and it may be possible to try and circumvent some of these, through a more rigorous and in-depth investigations.

In the following section, we further strengthen the main overall argument of this article – that simple abstractions, whilst useful, are limited, and that one needs to consider in detail the underlying biology if such work is to have long-term general significance. We support our argument by reviewing recent work on *natural genetic engineering*. This relatively novel – and still controversial – view of evolution, proposed mainly by Jim Shapiro, centres on the ability of individual cells to restructure their own genomic information in response to reproductive pressures. A deeper understanding of the fundamental underlying processes will benefit not only biologists attempting to gain new insights into evolution, but computer scientists and engineers seeking to use nature as the inspiration for robust and adaptable hard/software systems. Crucially, though, this investigation poses twin challenges to both biologists and computer scientists, and neither community will succeed in isolation. However, the anticipated benefits are wide-ranging and profound. As Shapiro himself argues: "These challenges should be high on the research agenda for the 21^{st} Century. It is likely that meeting them will lead us to new computing paradigms of great creative power, like evolution itself" [100].

5 NATURAL GENETIC ENGINEERING

Only relatively recently has the view of the genome changed from that of a collection of relatively independent genetic units, to that of a *system*, made up of an organised collection of interacting and interdependent modules [99]. The coding regions of genes provide the templates for proteins, which are generated when the gene is *expressed*, and these proteins can themselves affect the expression of other genes [61]. Shapiro argues that the genome has a precise architecture encoded by the distribution of various non-coding repetitive DNA sequences (often erroneously referred to as "junk DNA"). As this system-level architecture governs the "day to day" functioning of the cell, it

follows that alterations to this structure may well be much more significant in terms of evolution than modifications to individual proteins [99]. By "cutting and splicing" their own DNA, organisms may therefore reorganise both their repetitive and their coding sequences to generate new functional genomic systems. *Natural genetic engineering* (NGE) [98, 100] is the term given to this ability or capacity of organisms to modify or reorganise their own genome in response to certain pressures. As we have seen in Section 2.1, the shuffling of interdependent program modules is a very effective computational strategy when combined with some sort of selection pressure. This reorganisation may occur at different time-scales, and for a multitude of reasons. For example, it is clear that NGE occurs, at an intermediate time-scale, in the immune system [100]. Here, NGE progresses over the course of multicellular development, with cellular differentiation providing the system "clock". The notion of timescales, and the variety of them in the immune system, would seem to have been missed by the vast majority of AIS to date.

The problem of encoding an *extremely* large array of response molecules given a finite coding sequence region appears to have been solved by immune system lymphocytes utilising NGE. Here, sequences of controlled DNA rearrangements generate novel protein-coding regions that are used to generate new antigen-binding molecules. Moreover, a "real time" positive feedback loop amplifies the cells that have succeeded in generating molecules with a sufficient "fit" to the antigen, and then these cells undergo a further process of DNA "tweaking" to further increase specificity. This gives lymphocytes an extraordinary degree of responsiveness; they have evolved to *themselves* evolve rapid and specific adaptations [99, 100].

It is clear that NGE also occurs at a much more rapid time-scale than that of lymphocyte differentiation. Many organisms are capable of extremely rapid genomic rearrangement, perhaps the most striking example being that of the *ciliates*.

5.1 NGE in ciliates

Ciliate is a term applied to any member of a group of around 10,000 different types of single-celled organism that are characterized by two features: the possession of hair-like *cilia* for movement, and the presence of two kinds of *nuclei* instead of the usual one. One nucleus (the *micronucleus*) is used for sexual exchange of DNA, and the other (the *macronucleus*) is responsible for cell growth and proliferation. Crucially, the DNA in the micronucleus contains an "encoded" description of the DNA in the working macronucleus, which is decoded during development. This encoding "scrambles" fragments

of the functional genes in the macronucleus by both the permutation (and possible inversion) of partial *coding* sequences and the inclusion of non-coding sequences.

It is the macronucleus (that is, the "housekeeping" nucleus) that provides the RNA "blueprints" for the production of proteins. The micronucleus, on the other hand, is a dormant nucleus which is activated only during sexual reproduction, when at some point a micronucleus is converted into a macronucleus in a process known as *gene assembly*. During this process the micronuclear genome is converted into the macronuclear genome. This conversion reorganizes the genetic material in the micronucleus by removing noncoding sequences and placing coding sequences in their correct order. This "unscrambling" may be interpreted as a computational process.

The exact mechanism by which genes are unscrambled is not yet fully understood. We first describe experimental observations that have at least suggested possible mechanisms. We then describe a computational model of the process. We conclude this Section with a discussion of the computational and biological implications of this work.

5.2 Biological background

The macronucleus consists of millions of short DNA molecules that result from the conversion of the micronuclear DNA molecules. With few exceptions, each macronuclear molecule corresponds to an individual gene, varying in size between 400 b.p. (*base pairs*) and 15,000 b.p. (the average size is 2000 b.p.). The fragments of macronuclear DNA form a very small proportion of the micronucleus, as up to 98% of micronuclear DNA is noncoding, including intergenic "spacers" (that is, only $\sim 2\%$ of the micronucleus is coding DNA), and all noncoding DNA is excised during gene assembly.

IESs and MDSs

The process of decoding *individual* gene structures is therefore what interests us here. In the simplest case, micronuclear versions of macronuclear genes contain many short, noncoding sequences called *internal eliminated sequences*, or IESs. These are short sequences which, as their name suggests, are removed from genes and destroyed during gene assembly. They separate the micronuclear version of a gene into *macronuclear destined sequences*, or MDSs (Fig. 1a). When IESs are removed, the MDSs making up a gene are "glued" together to form the functional macronuclear sequence. In the simplest case, IESs are bordered on either side by pairs of identical repeat sequences (pointers) in the ends of the adjacent MDSs (Fig. 1b).

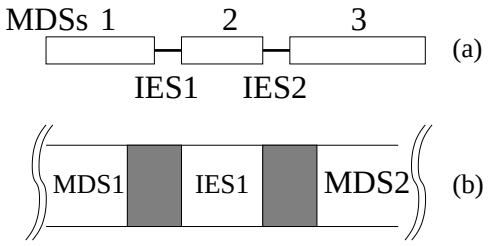

FIGURE 1
(a) Schematic representation of interruption of MDSs by IESs. (b) Repeat sequences in MDSs flanking an IES (the outgoing repeat sequence of MDS1 is equal to the incoming repeat sequence of MDS2)

Scrambled Genes

In some organisms, the gene assembly problem is complicated by the "scrambling" of MDSs within a particular gene. In this situation, the correct arrangement of MDSs in a macronuclear gene is present in a permuted form in the micronuclear DNA. For example, the actin I gene in *Oxytricha nova* is made up of 9 MDSs and 8 IESs, the MDSs being present in the micronucleus in the order 3–4–6–5–7–9–2–1–8, with MDS2 being inverted [95]. During the development of the macronucleus, the MDSs making up this gene are rearranged into the correct order at the *same* time as IES excision. Scrambling is often *further* complicated by the fact that some MDSs may be *inverted* (a 180° point rotation).

5.3 Fundamental Questions

Ciliates are remarkably successful organisms. The range of DNA manipulation and reorganization operations they perform has clearly been acquired during billions of years of evolution. However, some fundamental questions remain: what are the underlying molecular mechanisms of gene reconstruction and how did they evolve, and how do ciliates "know" which sequences to remove and which to keep?

Concerning the first question, Prescott proposes [92] that the "compression" of a working nucleus from a larger predecessor is part of a strategy to produce a "streamlined" nucleus in which "every sequence counts" (i.e., useless DNA is not present). This efficiency may be further enhanced by the dispersal of genes into individual molecules, rather than having them being

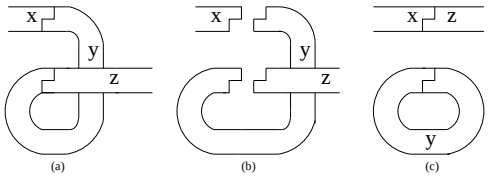

FIGURE 2
Excision

joined into chromosomes. However, so far we still know very little about the details and evolutionary origins of this intricate underlying molecular "machinery."

We may, perhaps, have more success in attempting to answer the second question: how are genes successfully reassembled from an encoded version? In the rest of this section we address this question from a computational perspective, and describe a computational model of the rearrangement process.

The model proposed by Prescott, Ehrenfeucht and Rozenberg (see, for example, [94]), is based on three intramolecular operations (that is, a single molecule folds on itself and swaps part of its sequence through recombination). The actual mechanics of cutting and splicing the DNA sequences are still not understood, but ciliates clearly contain the enzymatics tools (e.g., nucleases, ligases, etc.) needed to perform these tasks.

The first operation is the simplest, and is referred to as *loop, direct-repeat excision*. This operation deals with the situation depicted in Fig. 2, where two MDSs (x and z) in the correct (i.e., unscrambled) order are separated by an IES, y.

The operation proceeds as follows. The strand is folded into a loop with the two identical pointers aligned (Fig. 2a), and then staggered cuts are made (Fig. 2b). The pointers connecting the MDSs then join them together, while the IES self-anneals to yield a circular molecule (Fig. 2c).

The second operation is known as *hairpin, inverted repeat excision*, and is used in the situation where a pointer has two occurrences, one of which is inverted. The molecule folds into a hairpin structure (Fig. 3a) with the pointer and its inversion aligned, cuts are made (Fig. 3b) and the inverted sequence is reinserted (Fig. 3c), yielding a single molecule.

The third and final operation is *double-loop, alternating direct repeat exci-*

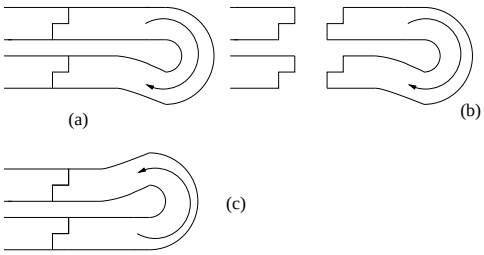

FIGURE 3
Inversion

sion/reinsertion. This operation is applicable in situations where two repeats of two pointers have interleaving occurrences on the same strand. The double loop folding is made such that the two pairs of identical pointer occurrences are aligned (Fig. 4a), cuts are made (Fig. 4b) and the recombination takes place, yielding the molecule from Fig. 4c.

The model has been successfully applied to all known experimental data on the assembly of real genes, including the actin I gene of *Urostyla grandis* and *Engelmanniella mobilis*, the gene encoding α telomere binding protein in several stichotrich species, and assembly of the gene encoding DNA polymerase α in *Sterkiella nova*. Descriptions of these applications are presented in [93]. From the perspective of the current article, the key aspect of this work is that the problem first originated in the study of a biological system. By expressing the operation of NGE in ciliates in terms of an abstract (but biologically plausible) topological operations, mathematicians were able to produce a computational model of the process that appears to account for every decrypted ciliate gene that has been observed to date. The feedback cycle is then complete when this abstract model is studied as a novel computational paradigm *in its own right* [28]. Biology and computer science are therefore inextricably tied together, as neither the computational model nor the descriptions it offers would be possible without tight interaction between the two disciplines.

6 CONCLUSIONS

The overriding message of this paper is that we feel that the bio-inspired computing in general, has reached an impasse. Through exploring two paradigms

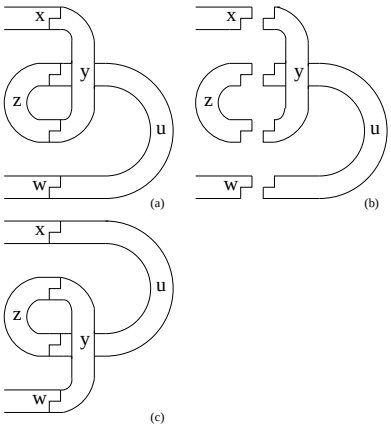

FIGURE 4
Excision/inversion

(Genetic Programming and Artificial Immune Systems), we have seen that, whilst some significant inroads have been made in the development of systems that in some way, mimic their natural counterpart, there still remains a wide gulf between that the artificial systems can do, compared with the natural systems. In both cases, we explored the limitations of each approach (a common theme being one of scaling), and concluded that it may be necessary, maybe essential, for each of those paradigms to revisit their biological roots, and take a look from whence they came. We then considered the areas of evolvable and evolved hardware systems. These systems, make use of techniques such as GP and AIS to evolve designs and configurations that can be placed into hardware systems, thus bringing benefits such as speed up and so on. We can conclude from this discussion that whilst it may seem appealing to place solutions into hardware (and in some cases necessary), again we meet the same problem as before, that of scale. Finally, we reviewed the area of Natural Genetic Engineering. Here, we showed how close interaction between biologists and computer scientists has generated a useful model of natural genetic engineering. This incredibly powerful framework for genomic rearrangement is one possible explanation of how organisms might "confront the issue of encoding infinity" [58]. We would say that the other areas of bio-inspired computing may take a lesson from the natural genetic engineering

community, and allow us to move forward to create the *second generation* of biocomputing systems.

REFERENCES

[1] http://www.cs.ucl.ac.uk/staff/W.Langdon/harrogate/BICS_Workshop_Report.htm.

[2] http://www.sigevo.org/.

[3] http://www.dangertheory.com.

[4] Lee Altenberg. (1994). Emergent phenomena in genetic programming. In Anthony V. Sebald and Lawrence J. Fogel, editors, *Evolutionary Programming — Proceedings of the Third Annual Conference*, pages 233–241, San Diego, CA, USA. World Scientific Publishing.

[5] Martyn Amos, editor. (2004). *Cellular Computing*. Series in Systems Biology. Oxford University Press, USA.

[6] Peter John Angeline. (1994). Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, pages 75–98. MIT Press.

[7] Modupe Ayara. (2005). *An immune inspired solution for adaptable error detection in embedded systems*. PhD thesis, University of Kent, Canterbury, Kent. UK.

[8] C.W.G.Lasarczykand W. Banzhaf. (2005). An algorithmic chemistry for genetic programming. In M. Keijzer, A. Tettamanzi, P. Collet, M. Tomassini, and J. van Hemert, editors, *Proc. 8th Europ. Conference on Genetic Programming, Lausanne, Switzerland, April 2005*, pages 1 – 12, Berlin. Springer Verlag.

[9] W. Banzhaf, P. Nordin, R. Keller, and F. Francone. (1998). *Genetic Programming - An Introduction*. Morgan Kaufmann, San Francisco, CA.

[10] Wolfgang Banzhaf. (1993). Genetic programming for pedestrians. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, page 628, San Francisco, USA. Morgan Kaufmann.

[11] Wolfgang Banzhaf. (2004). Artificial regulatory networks and genetic programming. In R. Riolo and B. Worzel, editors, *Genetic Programming - Theory and Applications*, pages 43 – 61. Kluwer Academic, Boston, MA, USA.

[12] Wolfgang Banzhaf and Julian Miller. (2004). The challenge of complexity. In A. Menon, editor, *Frontiers in Evolutionary Computation*, pages 243–260. Kluwer Academic, Boston, MA, USA.

[13] E. Ben-Jacob, I. Becker, Y. Shapira, and H. Levine. (2004). Bacterial linguistic communication and social intelligence. *Trends in Microbiology*, 12(8):366–372.

[14] P. Bentley, J. Greensmith, and S. Ujin. (2005). Two ways to grow artificial tissue. In C. Jacob, M. Pilat, P. Bentley, and J. Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems*, volume 3627 of *LNCS*, pages 139–152. Springer.

[15] Peter J. Bentley. (2004). Fractal proteins. *Genetic Programming and Evolvable Machines*, 5:71–101.

[16] H. Bersini. (1991). Immune network and adaptive control. In *Proceedings of the 1st European Conference on Artificial Life (ECAL)*, pages 217–226. MIT Press.

[17] H. Bersini. (1992). Reinforcement and recruitment learning for adaptive process control. In *Proc. Int. Fuzzy Association Conference (IFAC/IFIP/IMACS) on Artificial Intelligence in Real Time Control*, pages 331–337.

[18] H Bersini and F Varela. (1994). *The Immune Learning Mechansims: Recruitment, Reinforcement and their Applications*. Chapman Hall.

[19] Elena O. Budrene and Howard C. Berg. (1991). Complex patterns formed by motile cells of *Escherichia coli*. *Nature*, 349:630–633.

[20] Liu BZ and Deng GM. (1991). An improved mathematical model of hormone secretion in the hypothalamo-pituitary-gonadal axis in man. *J Theor Biol*, 150:51–8.

[21] X. Cai, S.L. Smith, and A.M. Tyrrell. (2005). Benefits of employing an implicit context representation on hardware geometry of cgp. In *Proceedings of International Conference on Evolvable Systems*, volume 3637 of *LNCS*, pages 143–154.

[22] R. O. Canham and A. M. Tyrrell. (September 2002). A multilayered immune system for hardware fault tolerance within an embryonic array. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems ICARIS*, pages 3–11, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

[23] L De Castro and F Von Zuben. (2000). The clonal selection algorithm with engineering applications. In *Proc. GECCO 2000 Workshop on Artificial Immune Systems*, pages 36–37. Morgan Kaufmann.

[24] D Cooke and J Hunt. (1995). Recognising Promoter Sequences using an Artificial Immune Systems. In *Proceedings of Intelligent Systems in Molecular Biology*, pages 89–97. AAAI Press.

[25] D. Dasgupta, editor. (1999). *Artificial Immune Systems and their Applications*. Springer.

[26] L. N. de Castro and J. Timmis. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer.

[27] A. Delgado. (2000). Control of nonlinear systems using a self-organizing neural network. *Neural Computing and Applications*, 9:113–123.

[28] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, and G. Rozenberg. (2004). *Computation in Living Cells: Gene Assembly in Ciliates*. Springer.

[29] M. Elowitz and S. Leibler. (January 2000). A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338.

[30] J. D. Farmer, N. H. Packard, and Perelson A. S. (1986). The immune system, adaptation, and machine learning. *Physica D*, 22:187–204.

[31] D.B. Fogel. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.

[32] L.J. Fogel, A.J. Owens, and M.J. Walsh. (1966). *Artificial Intelligenz through Simulated Evolution*. Wiley, New York.

[33] S Forrest, S Hofmeyr, and A Somayaji. (1997). Computer Immunology. *Communications of the ACM*, 40(10):88–96.

[34] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri. (1994). Self–nonself discrimination in a computer. In *Proceedings of the IEEE Symposium on Research Security and Privacy*, pages 202–212.

[35] S Forrest, A Perelson, L Allen, and R Cherukuri. (1994). Self-Nonself Discrimination in a Computer. In *Proc. of the IEEE Symposium on Research in Security and Privacy*, pages 202–212.

[36] T. Gardner, R. Cantor, and J. Collins. (January 2000). Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, 403:339–342.

[37] S. Garrett. (2005). How do we evaluate artificial immune systems? *Evolutionary Computation*, 13(2):145–177.

[38] S. M. Garrett. (2003). A paratope is not an epitope: Implications for clonal selection and immune networks. In *Proceedings of the 2nd International Conference on Artificial Immune Systems*, volume 2787.

[39] F. A. Gonzalez and D Dasgupta. (2003). Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines*, 4(4):383–403.

[40] D. Goodman, L Boggess, and A Watkins. (2002). Artificial immune system classification of multiple-class problems. In *Proc. of Intelligent Engineering Systems*, pages 179–184. ASME.

[41] D Goodman, L Boggess, and A Watkins. (2003). An investigation into the source of power for airs, an artificial immune classification system. In *Proc. of International Joint Conference on Neural Networks*, pages 1678–1683. IEEE.

[42] J. Greensmith, U. Aickelin, and S. Cayzer. (2005). Introducing Dendritic Cells as a Novel Immune-Inspired Algorithm for Anomaly Detection. In C Jacob, M Pilat, P Bentley, and J Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems*, volume 3627.

[43] L. Gritz and J.K. Hahn. (1995). Genetic programming for articulated figure motion. *J. of Visualization and Computer Animation*.

[44] Frederic Gruau. (17-21July 1993). Genetic synthesis of modular neural networks. In Stephanie Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 318–325, University of Illinois at Urbana-Champaign. Morgan Kaufmann.

[45] P. Haddow and G. Tufte. (1999). Evolving a robot controller in hardware. In *Proc. of the Norwegian Computer Science Conference*, pages 141–150.

[46] P. Haddow and G. Tufte. (2000). An evolvable hardware fpga for adaptive hardware. In *Proc. of the Congress on Evolutionary Computation*, pages 553–560.

[47] James V Hansen. (2003). Genetic programming experiments with standard and homologous crossover methods. *Genetic Programming and Evolvable Machines*, 4:53–66.

[48] E. Hart and J. Timmis. (2005). Application areas of ais: The past, the present and the future. In C. Jacob, M. Pilat, P. Bentley, and J. Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS 2005), LNCS 3627*, volume 3627 of *LNCS*, pages 126–138. Springer.

[49] Emma Hart and Peter Ross. (September 2002). Exploiting the analogy between immunology and sparse distributed memories: A system for clustering non-stationary data. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems ICARIS*, pages 49–58, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

[50] R. R. Hightower, S. A. Forrest, and A. S. Perelson. (1995). The evolution of emergent organization in immune system gene libraries. In L. J. Eshelman, editor, *Proc. 6th Int. Conf. Genetic Algorithms*, pages 344–350. Morgan Kaufmann.

[51] T. Higuchi, E. Takahashi, Y. Kasai, T. Itatani, M. Iwata, H. Sakanashi, M. Murakawa, I. Kajitani, and H. Nosato. (2003). *Computational Intelligence: The Experts Speak*, chapter Evolvable Hardware and Its Applications, pages 191–205.

[52] S Hofmeyr and S. Forrest. (2000). Architecture for an Artificial Immune System. *Evolutionary Computation*, 7(1):1289–1296.

[53] J. Holland. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

[54] G. Hollingworth, S. Smith, and A.M. Tyrrell. (2000). Safe intrinsic evolution of virtex devices. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardwar*, pages 195–202. IEEE Press.

[55] Gregory S. Hornby and Jordan B. Pollack. (2002). Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8:223–246.

[56] J Hunt and D Cooke. (1996). Learning using an artificial immune system. *Journal of Network and Computer Applications*, 19:189–212.

[57] J Hunt, J. Timmis, D Cooke, M Neal, and C King. (1998). *Artificial Immune Systems and their Applications*, chapter JISYS: Development of an Artificial Immune System for Real World Applications, pages 157–186. Springer.

[58] Information International Society for Complexity and Design, (February 2003). Transcript of discussion with J. Shapiro. http://www.iscid.org/james-shapiro-chat.php.

[59] M. Iwata, I. Kajtani, Y. Liu, N. Kajhara, and T. Higuchi. (2001). Implementation of a gate-level evolvable hardware chip,. In *In Proc of the 4th International Conference on Evolvable Systems*, LNCS, pages 38–49.

[60] C. Jacob, M. Pilat, P. Bentley, and J. Timmis, editors. (2005). *Proceedings of the 4th International Conference on Artificial Immune Systems (ICARIS 2005), LNCS 3627*. Springer.

[61] F. Jacob and J. Monod. (1961). Genetic regulatory mechanisms in the synthesis of proteins. *Journal of Molecular Biology*, 3:318–356.

[62] P. Kapranov and et al. (2002). Large-scale transcriptional activity in chromosomes 21 and 22. *Science*, 296:916–919.

[63] D. Keymeulen, M. Iwata, Y. Kuniyoshi, and T. Higuchi. (1999). Online evolution for a self-adaptive robot navigation system using evolvable hardware. *Artificial Life*, 4:359–393.

[64] D. Keymeulen, K. Konaka, and M. Iwata. (1997). Robot learning using gate level evolvable hardware. In *Proc. of the 6th European Workshop on Learning Robots*.

[65] J. Kim. (2002). *Integrating Artificial Immune Algorithms for Intrusion Detection*. PhD thesis, UCL, Department of Computer Science.

[66] T. Knight and J. Timmis. (2001). AINE: An immunological approach to data mining. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 297–304. IEEE Press.

[67] T. Knight and J. Timmis. (December 2003). A Multi-layerd Immune Inspired Machine Learning Algorithm. In A. Lotfi and M. Garibaldi, editors, *Applications and Science in Soft Computing*, pages 195–202. Springer.

[68] J. Koza. (1992). *Genetic Programming*. MIT Press, Cambridge, MA.

[69] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

[70] John R. Koza, Martin A. Keane, Jessen Yu, Forrest H Bennett III, and William Mydlowec. (2000). Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines*, 1:121–164.

[71] Renato A. Krohling, Yuchao Zhou, and Andy M. Tyrrell. (September 2002). Evolving fpga-based robot controllers using an evolutionary algorithm. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems ICARIS*, pages 41–46, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

[72] W. B Langdon. (2000). Size fair and homologous tree crossovers for tree genetic programming. *Genetic Programming and Evolvable Machines*, 1:95–119.

[73] William B. Langdon and Wolfgang Banzhaf. (2005). Repeated sequences in linear genetic programming genomes. *Complex sSstems*, 15.

[74] P. Layzell. (1998). A new research tool for intrinsic hardware evolution. In *Proc. of the 2nd International Conference on Evolvable Systems*, pages 47–56.

[75] JM Link and et al. (2005). Application of genetic programming to high-energy physics event selection. *Nuclear Instruments and Methods in Physics Research A*, 551:504–527.

[76] Y. Liu, M. Iwata, T. Higuchi, and D. Keymeulen. (2000). An integrated on-line learning system for evolving programmable logic array controllers. In *Proc. Parallel problem Solving from Nature*, pages 589–598.

[77] Jason Lohn, Gregory Hornby, and Derek Linden. (2004). Evolutionary antenna design for a NASA spacecraft. In Una-May O'Reilly, Tina Yu, Rick L. Riolo, and Bill Worzel, editors, *Genetic Programming Theory and Practice II*, pages 301 – 315. Kluwer, Boston, MA, USA.

[78] M.A. Lones and A.M. Tyrrell. (2002). Biomimetic representation with enzyme genetic programming. *Journal of Genetic Programming and Evolvable Machines*, 3(2):193–217.

[79] TE. McKee and T. Lensberg. (2002). Genetic programming and rough sets: A hybrid approach to bancruptcy classification. *European Journal on Operations Research*, 138:436–451.

[80] J. Miller and A. Thompson. (2000). Cartesian genetic programming. In *Third European Conf. Genetic Programming*, volume 1082 of *LNCS*.

[81] J. Miller and P. Thomson. (1998). Aspects of digital evolution: Evolvability and architecture. In *In Proc. of the Conf. on Parallel Problem Solving from Nature*, pages 927–936.

[82] Sibylle D. Muller, Jarno Marchetto, Stefano Airaghi, and Petros Koumoutsakos. (2002). Optimization based on bacterial chemotaxis. *IEEE Transactions on Evolutionary Computation*, 6(1):16–29.

[83] M. Neal. (September 2002). An artificial immune system for continuous analysis of time-varying data. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems ICARIS*, pages 76–85, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

[84] M Neal and J Timmis. (2004). *Recent Advances in Biologically Inspired Computing*, chapter : Once more unto the breach... towards artificial homeostasis? IGP.

[85] G Nicosia. (2004). *Immune Algorithms for Optimisation and Protien Structure Prediction*. PhD thesis, Department of Mathematics and Computer Science, University of Catania, Italy.

[86] Giuseppe Nicosia, Vincenzo Cutello, Peter J. Bentley, and Jon Timmis, editors. (2004). *Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS 2004), LNCS 3239*. Springer.

[87] P. Nordin. (1994). A compiling genetic programming system that directly manipulates the machine-code. In K. Kinnear, editor, *Advances in Genetic Programming*, pages 311 — 331. MIT Press, Cambridge, MA.

[88] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. (1996). Explicitly defined introns and destructive crossover in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 6, pages 111–134. MIT Press, Cambridge, MA, USA.

[89] A. S. Perelson. (1989). Immune network theory. *Immunological Review*, 110:5–36.

[90] Riccardo Poli, NicholasFreitag McPhee, and Jonathan Rowe. (2004). Exact schema theory and markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. *Genetic Programming and Evolvable Machines*, 5:31–70.

[91] Riccardo Poli, Christopher R. Stephens, Alden H. Wright, and Jonathan E. Rowe. (2002). On the search biases of homologuous crossover in linear genetic programming and variable-length genetic algorithms. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 868–876, New York. Morgan Kaufmann Publishers.

[92] David M. Prescott. (1998). Invention and mystery in hypotrich DNA. *Journal of Eukaryotic Microbiology*, 45(6):575–581.

[93] David M. Prescott and Grzegorz Rozenberg. (2004). Encrypted genes and their assembly in ciliates. In Amos [5].

[94] D.M. Prescott, A. Ehrenfeucht, and G. Rozenberg. (2001). Molecular operatons for DNA processing in hypotrichous ciliates. *European Journal of Protistology*, 37:241–260.

[95] D.M. Prescott and A.F. Greslin. (1992). Scrambled actin I gene in the micronucleus of *Oxytricha nova*. *Developmental Genetics*, 13:66–74.

[96] I. Rechenberg. (1994). *Evolutionsstrategie "93*. Frommann Verlag, Stuttgart.

[97] Michael A. Savageau. (2001). Design principles for elementary gene circuits: Elements, methods and examples. *Chaos*, 11(1):142–159.

[98] J.A. Shapiro. (1997). Genome organization, natural genetic engineering and adaptive mutation. *Trends in Genetics*, 13(3):98–104.

[99] James A. Shapiro. (1999). Genome system arhcitecture and natural genetic engineering. In L.F. Landweber and E. Winfree, editors, *DIMACS Workshop on Evolution as Computation*, pages 1–14. Springer.

[100] James A. Shapiro. (2005). Thinking about evolution in terms of cellular computing. In Martyn Amos and David A. Hodgson, editors, *Selected papers from the First International Symposium on Cellular Computing, December 9-10, 2004, Warwick, UK. Special issue of Natural Computing*. In press.

[101] S. Shirasuchi. (1996). Fpga as a key component for reconfigurable system. In *Proc. of the 1st International Conference on Evolvable Systems*, pages 23–32.

[102] Hans B. Sieburg and Oliver K. Clay. (1991). The cellular device machine development system for modeling biology on the computer. *Complex Systems*, 5:575–601.

[103] Karl Sims. (1994). Evolving 3D morphology and behaviour by competition. In R. Brooks and P. Maes, editors, *Artificial Life IV Proceedings*, pages 28–39, MIT, Cambridge, MA, USA. MIT Press.

[104] W.R. Smith. (1983). Qualitative mathematical models of endocrine systems. *Am. Journal of Physiology*, 245:473–477.

[105] E.D. Sontag. (2004). Some new directions in control theory inspired by systems biology. *IEEE Systems Biology*, 1:9–18.

[106] L. Spector and A. Robinson. (2002). Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3:7–40.

[107] S. Stepney, J.A. Clark, A. Tyrrell, C.G. Johnson, J. Timmis, D. Partridge, A. Adamatsky, and R.E. Smith. (May 2003). Journeys in non-classical computation: A grand challenge for computing research. Grand Challenge Report 7, National E-Science Centre, University of Edinburgh.

[108] S Stepney, R Smith, J Timmis, and A Tyrrell. (2004). Towards a Conceptual Framework for Artificial Immune Systems. In *LNCS 3239*, pages 53–64. Springer.

[109] S. Stepney, R. Smith, J. Timmis, A. Tyrrell, M. Neal, and A. Hone. (2005). Conceptual frameworks for artificial immune systems. *International Journal of Unconventional Computing*, 1(3):315–338.

[110] T. Stibor, K. M. Bayarou, and C. Eckert. (2004). An investigation of r-chunk detector generation on higher alphabets. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, pages 299–307. Springer.

[111] T. Stibor, P. Mohr, J. Timmis, and C. Eckert. (2005). Is negative selection appropriate for anomaly detection? In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 321–328. Springer.

[112] T. Stibor, J. Timmis, and C. Eckert. (2005). A comparative study of real-valued negative selection to statistical anomaly detection techniques. In C. Jacob, M. Pilat, P. Bentley, and J. Timmis, editors, *Proceedings of the 4th International Conference on Artificial Immune Systems*, volume 3627 of *LNCS*, pages 262–275.

[113] K.C. Tan, C.M. Chew, K.K. Tan, L.F. Wang, and Y.J. Chen. (2002). Autonomous robot navigation via intrinsic evolution. In *Proc. of the Congress on Evolutionary Computation, Part of the IEEE World Congress on Computational Intelligence*, pages 1272–1277. IEEE Press.

[114] Astro Teller and Manuela Veloso. (1996). PADO: A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press.

[115] A. Thompson. (1995). Evolving electronic robot controllers that exploit hardware resource. In *Proc. of the 3rd European Conf. on Artificial Life*, pages 640–656.

[116] A. Thompson. (1996). An evolved circuit, intrinsic in silicon, entwined with physics. In *In Proc. of the 1st International Conference on Evolvable Systems*, LNCS, pages 390–405.

[117] J Timmis. (2000). *Artificial Immune Systems: a novel data analysis technique inspired by the immune system PhD Thesis*. University of Wales, Aberystwyth.

[118] J. Timmis and P. J. Bentley, editors. (2002). *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS 2002)*. University of Kent Printing Unit.

[119] J. Timmis and T. Knight. (2001). *Data Mining: A Heuristic Approach*, chapter Artificial immune systems: Using the immune system as inspiration for data mining., pages 209–230. Idea Group.

[120] J. Timmis and M. Neal. (2001). A Resource Limited Artificial Immune System. *Knowledge Based Systems*, 14(3/4):121–130.

[121] J. Timmis and M. J. Neal. (December 2000). A Resource Limited Artificial Immune System for Data Analysis. *Research and Development in Intelligent Systems XVII*, pages 19–32. Proceedings of ES2000, Cambridge, UK.

[122] Jon Timmis, Peter Bentley, and Emma Hart, editors. (2003). *Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS 2003), LNCS 2787*. Springer.

[123] F Varela, A Countinho, B Dupire, and N Vaz. (1988). Cognitive Networks: Immune, Neural and Otherwise. *Theoretical Immunology*, 2:359–375.

[124] J. von Neumann. (1967). In L. Jeffress, editor, *Cerebral Mechanisms of Behavior*, page 1. Hafter.

[125] A Watkins. (2001). *An Artificial Immune Recognition System*. Mississippi State University. MSc Thesis.

[126] A. Watkins. (2005). *Exploiting Immunological Metaphors in the Development of Serial, Parallel and Distributed Learning Algorithms*. PhD thesis, University of Kent, Computing Laboratory.

[127] A. Watkins, B. Xintong, and A. Phadke. (2003). Parallelizing an immune-inspired algorithm for efficient pattern recognition. In *Intelligent Engineering Systems through Artificial Neural Networks: Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Complex Systems and Artificial Life*, pages 224–230. ASME Press.

[128] Andrew Watkins, Jon Timmis, and Lois Boggess. (2004). Artificial immune recognition system (AIRS): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, 5(1):291–317.

[129] Ian Weaver and et al. (2003). Identification and characterization of multi-species conserved sequences. *Genome Research*, 13:2507–2518.

[130] Ian Weaver and et al. (2004). Epigenetic programming by maternal behavior. *Nature Neuroscience*, 7:847–854.

[131] R. Weiss, G. Homsy, and T.F. Knight Jr. (1999). Toward *in-vivo* digital circuits. In L.F. Landweber and E. Winfree, editors, *DIMACS Workshop on Evolution as Computation*, pages 275–295. Springer.

[132] S. Wierzchon and U. Kuzelewska. (September 2002). Stable clusters formation in an artificial immune system. In Jonathan Timmis and Peter J. Bentley, editors, *Proceedings of the 1st International Conference on Artificial Immune Systems ICARIS*, pages 68–75, University of Kent at Canterbury. University of Kent at Canterbury Printing Unit.

[133] A.H.C. Wong, I.I. Gottesman, and A. Petronis. (2005). Phenotypic difference in genetically identical organisms: the epigenetic perspective. *Human Molecular Genetics*, 14:R11–R18.

[134] R.S. Zebulum, M.M.R. Vellasco, and M.A.C. Pacheco. (2001). *Evolutionary Electronics: Automatic Design of Electronic Circuits*. CRC Press.