

Artificial Immune Systems and the Grand Challenge for Non-classical Computation

Susan Stepney¹, John A. Clark¹, Colin G. Johnson²,
Derek Partridge³, and Robert E. Smith⁴

¹ Department of Computer Science, University of York

² Computing Laboratory, University of Kent

³ Department of Computer Science, University of Exeter

⁴ The Intelligent Computer Systems Centre, University of the West of England

Abstract. The UK Grand Challenges for Computing Research is an initiative to map out certain key areas that could be used to help drive research over the next 10–15 years. One of the identified Grand Challenges is *Non-Classical Computation*, which examines many of the fundamental assumptions of Computer Science, and asks what would result if they were systematically broken. In this discussion paper, we explain how the sub-discipline of Artificial Immune Systems sits squarely in the province of this particular Grand Challenge, and we identify certain key questions.

The UK Grand Challenges in Computing Research

The UK Computing Research Committee (UKCRC) is organising *Grand Challenges for Computing Research*, to discuss possibilities and opportunities for the advancement of computing science. It has asked the UK computing research community to identify ambitious, long-term research initiatives. The original call resulted in 109 submissions, which were discussed at a workshop in late 2002, and refined into a handful of composite proposals. (Further details, and how to get involved, can be found at [16]).

In this discussion paper, we explain how Artificial Immune Systems (AIS) sit squarely in the province of one of the resulting composite proposals: *Journeys in Non-Classical Computation*. First we summarise the Challenge itself [16], and then show where the sub-discipline of AIS falls within its remit, and the particular questions AIS raise and address. AIS provide excellent exemplars of non-classical computational paradigms, and provide a resource for studying *emergence*, necessary for a full science of complex adaptive systems.

We hope this discussion paper will encourage the AIS community to position and exploit their research within the wider Non-Classical Computation arena.

The Grand Challenge of Non-classical Computation

Some events permit huge increases in kinds and levels of complexity; these are termed *gateway events* [7]. Such events open up whole new kinds of phase space to a

system's dynamics. Gateway events during evolution of life on earth include the appearance of eukaryotes (organisms with a cell nucleus), an oxygen atmosphere, multi-cellular organisms, and grass. Gateway events during the development of mathematics include each invention of a new class of numbers (negative, irrational, imaginary, ...), and dropping Euclid's parallel postulate.

A gateway event produces a profound and fundamental change to the system: once through the gateway, life is never the same again. We are currently poised on the threshold of a significant gateway event in computation: that of breaking free from many of our current "classical computational" assumptions. The corresponding Grand Challenge for computer science is **to journey through the gateway event obtained by breaking our current classical computational assumptions, and thereby develop a mature science of Non-Classical Computation.**

Journeys versus Goals

Many Grand Challenges are cast in terms of goals, of end points: "achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to earth" [11], mapping the human genome, proving whether $P = NP$ or not. We believe that a goal is not the best metaphor to use for this particular Grand Challenge, however, and prefer the metaphor of a *journey*.

The journey metaphor emphasises the importance of the entire process, rather than emphasising the end point. In the 17th and 18th centuries it was traditional for certain sections of "polite society" to go on "a Grand Tour of Europe", spending several years broadening their horizons: the experience of the entire journey was important. And in the Journey of Life, death is certainly not the goal! Indeed, an open journey, passing through gateway events, exploring new lands with ever expanding horizons, need not have an end point.

Journeys and goals have rather different properties. A goal is a fixed target, and influences the route taken to it. With an open journey of exploration, however, we are not able to predict what will happen: the purpose of the journey is discovery, and our discoveries along our journey will suggest new directions for us to take. We can suggest starting steps, and some intermediate way points, but not the detailed progress, and certainly not the end result.

Thinking of the Non-Classical Computation Challenge in terms of a journey, or rather several journeys, of exploration, we can today spy some early *way points*, as we peer through the gateway. As the community's journey progresses, new way points will heave into view, and we can alter our course to encounter these as appropriate.

Six Classical Paradigms to Disbelieve before Breakfast

Classical computing is an extraordinary success story. But there is a growing appreciation that it encompasses only a tiny subset of all computational possibilities.

Discoveries may emerge when an assumption that *this has to be the case* is found merely to be an instance of *this has always been the case*, and is changed. We wish to enable such discoveries by highlighting several assumptions that define classical computing, but that are not necessarily true in all computing paradigms. Researchers

in many different fields are already challenging some these (for example, [8]), and we encourage the community to challenge more, in whatever ways seem interesting. In later sections we discuss alternatives in more detail. (Some of the categories arguably overlap.)

1 The Turing Paradigm

classical physics: states have particular values, information can be freely copied, information is local. Rather, at the quantum level states may exist in superpositions, information cannot be cloned, and entanglement implies non-locality.

atomicity: computation is discrete in time and space; there is a before state, an after state and an operation that transforms the former into the latter. Rather, the underlying implementation realises intermediate physical states.

unbounded resources: Turing machines have infinite tape state, and zero power consumption. Rather, resources are always constrained.

substrate as implementation detail: the machine is logical, not physical. Rather, a physical implementation of one form or another is always required, and the particular choice has consequences.

universality is a good thing: one size of digital computer, one size of algorithm, fits all problems. Rather, a choice of implementation to match the problem, or hybrid solutions, can give more effective results.

closed and ergodic systems: the state space is pre-determined. Rather, the progress of an interactive computation may open up new regions of state space in a contingent manner.

2 The von Neumann Paradigm

sequential program execution. Rather, parallel implementations already exist.

fetch-execute-store model of program execution. Rather, other architectures already exist, for example, neural nets, FPGAs.

the static program: the program stays put and the data comes to it. Rather, the data could stay put and the processing rove over it.

3 The Output Paradigm

a program is a black box: it is an oracle abstracted away from any internal structure. Rather, the computation's *trajectory* can be as interesting, or more interesting, than its final result.

a program has a single well-defined output channel. Rather, we can chose to observe other aspects of the physical system as it executes.

a program is a mathematical function: logically equivalent systems are indistinguishable. Rather, correlations of multiple outputs from different executions, or different systems, may be of interest.

4 The Algorithmic Paradigm

a program maps the initial input to the final output, ignoring the external world while it executes. Rather, a system may be an ongoing adaptive process, with inputs provided over time, with values dependent on how it interacts with the open unpredictable environment; identical inputs may provide different outputs, as the system learns and adapts to its history of interactions; there is no prespecified endpoint.

the computer can be switched on and off: computations are bounded in time, outside which the computer does not need to be active. Rather, the computer may engage in a continuous interactive dialogue, with users and other computers.

randomness is noise is bad: most computer science is deterministic. Rather, nature-inspired processes, in which randomness or chaos is essential, are known to work well.

5 The Refinement Paradigm

incremental transformational steps move a specification to an implementation that realises that specification. Rather, there may be a discontinuity between specification and implementation, for example, bio-inspired recognisers.

binary is good: answers are crisp yes/no, true/false, and provably correct. Rather, probabilistic, approximate, and fuzzy solutions can be just as useful, and more efficient.

a specification exists, either before the development and forms its basis, or at least after the development. Rather, the specification may be an emergent and changing property of the system, as the history of interaction with the environment grows.

emergence is undesired, because the specification captures everything required, and the refinement process is top-down. Rather, as systems grow more complex, this refinement paradigm is infeasible, and emergent properties become an important means of engineering desired behaviour.

6 The “Computer as Artefact” Paradigm

computation is performed by artefacts: computation is not part of the real world. Rather, in some cases, nature “just does it”, for example, optical Fourier transforms.

the hardware exists unchanged throughout the computation. Rather, new hardware can appear as the computation proceeds, for example, by the addition of new resources. Also, hardware can be “consumed”, for example, a chemical computer consuming its initial reagents. In the extreme, nanites will construct the computer as part of the computation, and disassemble it at the end.

the computer must be on to work. Rather, recent quantum computation results [9] suggest that you don't even need to "run" the computer to get a result!

Doubtless there are other classical paradigms that we accept almost without question. They too can be fruitfully disbelieved.

There is a gulf between the maturity of classical computing and that of the emerging non-classical paradigms. For classical computing, intellectual investment over many years is turning craft into science. To fully exploit emerging non-classical computational approaches we must seek for them such rigour and engineering discipline as is possible. What that science will look like is currently unclear, and the Grand Challenge encourages exploration.

The Real World: Breaking the Turing Paradigm

Real World as Its Own Computer

The universe doesn't need to compute, it just does it. We can take the *computational stance*, and view many physical, chemical and biological processes *as if* they were computations: the Principle of Least Action "computes" the shortest path for light; water "computes" its own level; evolution "computes" fitter organisms (evolution of bacterial resistance is evolution of real bacteria against an increasingly strong "data set" of attacks to the bacterial "population"); DNA and morphogenesis "computes" phenotypes; the immune system "computes" antigen recognition.

This natural computation can be more effective than a digital simulation. For example, the real world performs quantum mechanical computations exponentially faster than can classical simulations [6].

Real World as Our Computer

Taking the computational stance, we may exploit the way the world works to perform "computations" for us. We set up the situation so that the natural behaviour of the real world gives the desired result.

There are various forms of real world sorting and searching, for example. Centrifuges exploit differences in density to separate mixtures of substances, a form of gravitational sorting. Vapours of a boiling mixture are richer in the components that have lower boiling points; distillation exploits this to give a form of thermal sorting. Chromatography provides a chemical means of separation. Other kinds of computations exist: for example, optics performs Fourier transforms.

Real World as Analogue Computer

We may exploit the real world in more indirect ways. The "computations" of the "real world as our computer" are very direct, yet often we are concerned with more abstract questions. Sometimes we can harness the physical world to provide the re-

sults that we need: we may be able to set up the situation so that there is an *analogy* between the computation performed by the real world, and the result we want.

There is an age-old mechanism for finding the longest stick of spaghetti in an unruly pile, exploiting the physics of gravity and rigidity: we can use this to sort by setting up an analogy between spaghetti strand length and the quantity of interest.

Classical computing already exploits physics at the level of electron movements. But there are other ways of exploiting nature.

Analogue computing itself exploits the properties of electrical circuits as analogues of differential equations (amongst other analogues).

DNA computing encodes problems and solutions as sequences of bases (strands) and seeks to exploit mechanisms such as strand splitting, recombination and reproduction to perform calculations of interest. This can result in vast parallelism, of the order of 10^{20} strands.

Quantum computing presents one of the most exciting developments for computer science in recent times, breaking out of the classical Turing paradigm. As its name suggests, it is based on quantum physics, and can perform computations that cannot be effectively implemented on a classical Turing machine. It exploits interference, many worlds, entanglement and non-locality. Newer work still is further breaking out of the binary mind-set, with multiple-valued *qudits*, and continuous variables.

Real World as Inspiration

Many important techniques in computer science have resulted from simulating (currently very simplified) aspects of the real world. *Meta-heuristic search* techniques draw inspiration from many areas, including physics (simulated annealing), evolution (genetic algorithms, genetic programming), neurology (artificial neural networks), immunology (artificial immune systems [5]), and social networks (ant colony optimisation [2]).

In the virtual worlds inside the computer, we are no longer constrained by the laws of nature. Our simulations can be other than the way the real world works. For example, we can introduce novel evolutionary operators to our genetic algorithms, novel kinds of selection algorithms to our artificial immune systems, and even, as we come to understand the embracing concepts, novel kinds of complex adaptive systems themselves. The real world is our inspiration, not a restriction.

Reality based computing techniques have proved successful, or look promising, yet the science underpinning their use comes nowhere near matching the science of classical computing. Given a raft of nature-inspired techniques we would like to get from problem to solution efficiently and effectively, and we would like to reason about the performance of the resulting systems. But this falls outside the classical refinement paradigm.

Massive Parallelism: Breaking the von Neumann Paradigm

Parallel processing (Cellular Automata, etc) and other non-classical architectures break out of the sequential, von Neumann, paradigm.

Under the classical computational assumptions, any parallel computation can be serialised, yet parallelism has its advantages.

Real-time response to the environment. The environment evolves at its own rate, and a single processor might not be able to keep pace. (Possibly the ultimate example of this will be the use of vast numbers of nanotechnological assemblers (nanites) to build macroscopic artefacts. A single nanite would take too long, by very many orders of magnitude.)

Better mapping of the computation to the problem structure. The real world is intrinsically parallel, and serialisation of its interactions to map to the computational structure can be difficult. Parallelism also permits colocation of each hardware processor and the part of the environment with which it interacts most. It then permits colocation of the software: software agents can roam around the distributed system looking for the data of interest, and meeting other agents in a context dependent manner. For example, artificial antibodies can patrol the network they are protecting.

And once the classical computational assumptions are challenged, we can see that serialisation is not necessarily equivalent.

Fault tolerance. Computation requires physical implementation, and that implementation might fail. A parallel implementation can be engineered to continue working even though some subset of its processors have failed. A sequential implementation has only the one processor.

Interference/interaction between devices. Computation requires physical implementation, and those implementations have extra-logical properties, such as power consumption, or electromagnetic emissions, which may be interpreted as computations in their own right (see later). These properties may interfere when the devices are running in parallel, leading to effects not present in a serialised implementation. (Possibly the ultimate example of this is the exponentially large state space provided by the superposed parallel qubits in a quantum computer.)

The use of massive parallelism introduces new problems. The main one is the requirement for decentralised control. It is just not possible for a single centralised source to exercise precise control over vast numbers of heterogeneous devices. Part of this issue is tackled by the sister Grand Challenges in *Ubiquitous Systems* [16], and part is addressed in the later section, on open processes.

In the Eye of the Beholder: Breaking the Output Paradigm

The classical paradigm of program execution is that an abstract computation processes an input to produce an output. This input-output mapping is a logical property of the computation, and is all that is important: no intermediate states are of interest, the computation is independent of physical realisation, and different instances of the computation yield precisely the same results.

Computation, however, is in the eye of the beholder. Algorithms are implemented by physical devices, intermediate states exist, physical changes happen in the world, different devices are distinguishable. Any information that can be observed in this physical world may be used to enrich the perceived computation [4].

Logical Trajectory Observations. An executing algorithm follows a trajectory through its logical state space. (Caveat: this is a classical argument: intermediate quantum computational states may be in principle unobservable.) Typically, this trajectory is not observed (except possibly during debugging). This is shockingly wasteful: such logical information can be a computational resource in its own right. For example, during certain types of heuristic search the trajectory followed can give more information about a sought solution than the final “result” of the search itself.

Physical Trajectory Observations. An executing algorithm is accompanied by physical changes to the world: for example, it consumes trajectory-dependent power as it progresses, and can take trajectory-dependent time to complete. Such physical resource consumption can be observed and exploited as a computational resource, for example, to deduce features of the logical trajectory. (For example, recent attacks on smart cards have observed such things to deduce secret key information [3].)

Differential Observations. An executing algorithm is realised in a physical device. Physical devices have physical characteristics that can change depending on environmental conditions such as temperature, and that differ subtly across logically identical devices. So one can observe not merely the output of a single execution, but a set of outputs from a family of executions, from multiple systems, from different but related systems, and perform differential analyses.

Higher-Order Observations. These are observations not of the program execution itself, but of the execution of the program used to design (the program used to design...) the program.

Open Processes: Breaking the Algorithmic Paradigm

In the classical paradigm, the ultimate goal of a computation is reaching a fixed point: the final output, the “result” of the computation, after which we may switch off the computer. The majority of classical science is also based around the notion of fixed-point equilibrium and ergodicity (the property that a system has well defined spatial and temporal averages, as any state of the system recurs with non-zero probability).

Many modern scientific theories consider systems that lack repetition and stability: they are *far-from-equilibrium* and *non-ergodic*. The most obvious non-ergodic, far from equilibrium system is life itself, characterised by perpetual evolution (change).

Consider the most basic of chaotic systems: the logistic process:

$$x_{t+1} = Rx_t(1 - x_t)$$

The behaviours of various logistic processes as a function of the parameter R form the well-know bifurcating, chaotic *logistic curve* (see, for example, figure 21 of [12]). For small values of R , these logistic processes have a fixed point attractor. As R increases, the attractor becomes period two, then four, then eight. This *period doubling* continues, and the values of R where each doubling occurs get closer together. For $R > 3.569945671\dots$ the logistic process’s attractor goes through an infinite number of values (except for a few “islands” of order, of attractors with multiples of odd periods). There is a phase transition from order (the region of period doubling) to chaos

(“random” behaviour). The phase transition point at $R = 3.569945671\dots$ is the so-called *edge of chaos*.

Consider a discretised process whose underlying (continuous) dynamics are those of the logistic equation, and imagine taking samples of length L bits. Construct an automaton machine that represents the process, for a given L . There is a clear phase transition (a peak in the machine size versus the entropy of the bit sequence) as we move from the period doubling region to the chaotic region. At the phase transition, the machine size versus the length of the sequence L *expands without bound*. That is, at the edge of chaos, the logistic machine requires an infinite memory machine for accurate representation: there is a leap in the level of intrinsic computation going on.

At the edge of chaos, we can add new resources (computational or physical) and get results that are neither redundant (as in the structured period doubling regime) nor random (as in the chaotic regime). Within the classical paradigm, such conditions would be anathema, indicating unceasing variety that never yields “the solution”. But in life-like systems, there is simultaneously sustained order, and useful innovation. New matter can be brought into such systems and used in ways that are neither redundant nor random. In this setting, emergence of the unforeseen is a desirable property, rather disruptive noise.

Computing often attempts to exploit the biological paradigm: cellular automata, evolutionary computation, recurrent networks (autocatalytic, neural, genomic, cytokine immune system, ecological webs, ...), social insect and agent-based systems, DNA-computing, and nanite-systems that build themselves. The implementations in most of these cases, however, are locked into themselves, *closed*, unable to take on new matter or information, thus unable to truly exploit emergence.

Open systems are systems where new resources, and new *kinds* of resources, can be added at any time, by external agency, or by the actions of the system. (For example, an immune system might have inoculation material introduced by an external agent, or evolve a new kind of immune cell.) These new resources can provide *gateway events*, that fundamentally alter the character of the system dynamics, by opening up new kinds of regions of phase space, allowing new possibilities.

Computational systems are beginning to open themselves to unceasing flows of information (if not so much to new matter). The openness arises, for example, through human interactivity as a continuing dialogue between user and machine [15], through unbounded networks, through robotic systems with energy autonomy. As computers become ubiquitous, the importance of *open systems physics* to understanding computation becomes crucial.

Artificial Immune Systems, and the Grand Challenge

The Inspiration and the Analogy

AIS are relatively recent example of using the real world as computational inspiration. One such inspiration for AIS runs something like: the vertebrate immune system fights infection by recognising and attacking *non-self*; can an analogous computational system be used to detect, diagnose, and fight computer intrusions (from hacker attacks to computer viruses)?

In addition to this obvious security metaphor, many other AIS application areas have been developed, to cover more general *anomaly detection*, optimisation, fault tolerance, and general purpose machine learning applications such as recognition and classification.

The Models

There are two main classes of models for AIS: the population-based (or selection) model, and the network model (see [5] for details), which have impacts on different areas of the main Challenge.

The Population-Based Model

The population-based model is computationally inspired by the processes during early maturation of immune cells, before they are released into the lymphatic system. It uses some particular algorithm (positive, negative, clonal, ...) to select a set of *recognisers* (supervised learning) or *classifiers* (unsupervised learning), of self or non-self (details depending on the precise algorithm).

This model fits well with the other bio-inspired soft learning systems, such as neural nets and genetic algorithms. The major contributions to the Grand Challenge are in the area of *breaking the refinement paradigm*.

In all these soft learning approaches, there is a discontinuity between the problem statement and the bio-inspired solution. With both NNs and AISs, the solution is distributed over the entire system. Each artificial antibody may recognise several different antigens: the specific response to a particular antigen is a global property of all the antibodies. The complex response emerges from the simpler behaviour of individual parts.

The way point questions specific to AIS include:

- What are the effect of aspects and parameters of the selection algorithm on the outcome and applicability of the algorithms?
- Can we observe the computational trajectory taken during selection and recognition to get useful information?

The immune system population-based model forms an excellent exemplar for breaking the refinement paradigm. The challenge is to develop a *science of non-classical refinement*, that permits quantitative reasoning about *all* bio-inspired algorithms, including AISs, in both a bottom up and top down manner:

- *understanding* and *predicting* the global recognisers and classifiers that emerge from a collection of local non-specific agents
- a means to *design* and *implement* appropriate sets of recognisers or classifiers for particular applications, in a rigorous (but possibly non-incremental) way
- *quantitative description methods* that enable rigorous reasoning about the behaviour of AISs, such that they can be used reliably in critical applications

Taking inspiration and input from all the bio-inspired learning algorithms, major way points on the Non-Classical Computation journey are

- a *general theory of learning systems* that includes neural, evolutionary, and immune systems as special cases
- use of the general theory to develop *more effective kinds* of learning systems, inspired by, but not based upon, any known biological processes

The Network Model

The immune system network model is computationally inspired by the biological processes used to maintain a dynamic “memory” of immune responses, in a system where the lifetime of individual immune memory cells is on the order of weeks, yet the memory itself persists on the order of years or decades.

The immune system network model forms a superb exemplar for *breaking the output paradigm*. It is one of many dynamic network models that occur in biological and social systems, from Kauffman’s autocatalytic networks [10], and genomic control networks, through dynamical models of neural networks, to ecological food webs, and social and technological networks. All these subject areas could benefit from better networks models. Much of the existing mathematical network theory is restricted to static, homogeneous, structured, closed networks, since these are the simplest, most tractable models to work with. However, these are not realistic models of biological networks. Antibodies rove around the body (network, system, ...) looking for the anomalies, and new kinds of attacks call for new kinds of defence. The challenge is to develop a pragmatic theory of *dynamic, heterogeneous, unstructured, open networks* [13].

- *Dynamic*: the network is not in steady state or equilibrium, but is far from equilibrium, governed by attractors and trajectories. (*Swarm networks* may offer insights to this kind of dynamics [2])
- *Heterogeneous*: the nodes, the connections, and the communications can be of many different types, including higher order types.
- *Unstructured*: the network connectivity has no particular regularity: it is not fully regular, or fully connected, or even fully random. Clearly there need to be *some* kinds of regularity present, but these are likely to be of kinds that cannot be reasoned about in terms of simple averages or mean field notions; they are more likely have fractal structure. Some recent advances in *Small World* networks offer intriguing new insights [1][14].
- *Open (metadynamic)*: the structures are unbounded, and the components are not fixed: nodes and connections may come and go; new *kinds* of nodes and connections may appear.

A general theory of such networks would have application well beyond AISs. Such a theory is a basic requirement of complex systems development in general, one application of which is pervasive, or ubiquitous, computing (the subject of another Grand Challenge [16]). Such a theory a necessary way point for answering such challenging questions as

- *Computation as a dynamical process*. What are the various attractors of a dynamical computation? How can we encourage the system to move to a “better” attractor? How can we map the route through intermediate attractors that it should take?

- *Computation at the edge of chaos.* What are its capabilities? How can we hold a system at the edge, far from equilibrium, to perform useful computations? How can we make it self-organise to the edge?
- *Designed emergence.* How can we design (refine) open systems that have desired emergent properties? And do not have undesired emergent properties?
- *Open systems science.* What are the fundamental properties of open systems? How can we predict the effect of interventions (adding new things, or removing things) to the system? How can we understand the effect of a gateway event that opens up new kinds of regions of phase space to a computation? How can we design a system such that gateway events, natural changes to phase space, can occur endogenously?

The Biological Models

Like many biologically inspired computational ideas, the computer science and the biology are developing in parallel. The natural immune system, in particular, is an exceedingly complicated and not well understood biological mechanism. The current discipline of AIS may have been *inspired* by the biology, but it is painfully clear that AISs are but a pale shadow of the vast complexity and subtlety of the natural immune system. Computer scientists, mathematicians, and immunologists working together can ask, and answer, some deep and interesting questions. For example:

- How might we use the real immune system, and other real physical and biological systems, for computation?
- To what extent is the working of the immune system, and other biological systems, dictated by the physical substrate? Can all putative “immune” responses be realised on all substrates? Do some diseases exploit *computational* constraints of the immune system to defeat it?
- How can we use models to decide which parts of the biology are necessary for correct robust functioning, which parts are necessary only because of the particular physical realisation, and which parts merely contingent evolutionary aspects?
- How can we use nature inspired computation to build “better than reality” systems? What are the computational limits to what we can simulate?

Conclusions

AIS do not break all the classic computational paradigms: for example, they do not (yet?) use concepts from quantum physics. However, they do challenge some of the major paradigms. The population-based model is a good exemplar for examining alternatives to the refinement paradigm, and the network model is an excellent exemplar for examining open network dynamics and emergence, necessary for a full science of complex adaptive systems.

Classical physics did not disappear when modern physics came along: rather its restrictions and domains of applicability were made explicit. Similarly, the various forms of non-classical computation will not supersede classical computation: they will augment and enrich it. And when a wide range of tools is available, we can pick the best one, or the best combination, for each job. For example, it might be that

using a quantum algorithm to reduce a search space, and then a meta-heuristic search to explore that, is more effective than using either algorithm alone. AISs form one of a whole array of novel approaches to computation that are becoming available. It is important that these separate areas are not seen as independent. Rather, their results and insights should provide valuable groundwork for the overarching challenge, to produce a **fully mature science of all forms of computation, that unifies the classical and non-classical paradigms.**

References

1. A.-L. Barabasi. *Linked: the new science of networks*. Perseus, 2002.
2. E. W. Bonabeau, M. Dorigo, G. Theraulaz. *Swarm Intelligence*. OUP, 1999
3. S. Chari, C. S. Jutla, J. R. Rao, P. Rohatgi. *Power analysis*. In A. McIver, C. Morgan, eds. *Programming Methodology*. Springer, 2003.
4. J. A. Clark, S. Stepney, H. Chivers. Breaking the model (submitted)
5. L. N. de Castro, J. Timmis. *Artificial Immune Systems*. Springer, 2002.
6. R. P. Feynman. Simulating Physics with Computers. *Int. J. Theor. Phys.* 21(6/7) 1982.
7. M. Gell-Mann. *The Quark and the Jaguar*. Abacus, 1994.
8. T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, T. Pellizzari. *Non-Standard Computation*. Wiley, 1998.
9. R. Jozsa: Characterising Classes of Functions Computable by Quantum Parallelism. *Proc. R. Soc. Lond. A* 435, 1991.
10. S. A. Kauffman. *The Origins of Order*. OUP, 1993.
11. J. F. Kennedy. Announcement to the US Congress. 25 May, 1961.
12. H.-O. Peitgen, P. H. Richter. *The Beauty of Fractals*. Springer, 1986.
13. S. Stepney. Critical Critical Systems. In *Formal Aspects of Security, FASEC'02*. LNCS vol 2629, Springer, 2003.
14. D. J. Watts. *Small Worlds*. Princeton University Press, 1999.
15. P. Wegner. Why interaction is more powerful than algorithms. *CACM*, 40(5) 1997.
16. UK Grand Challenges in Computing Research Website.
http://umbriel.dcs.gla.ac.uk/NeSC/general/esi/events/Grand_Challenges