Reservoir Computing in *Materio*

Matthew Nicholas Dale

PhD

University of York

Computer Science

September 2018

Abstract

Reservoir Computing first emerged as an efficient mechanism for training recurrent neural networks and later evolved into a general theoretical model for dynamical systems. By applying only a simple training mechanism many physical systems have become exploitable unconventional computers. However, at present, many of these systems require careful selection and tuning by hand to produce usable or optimal reservoir computers. In this thesis we show the first steps to applying the reservoir model as a simple computational layer to extract exploitable information from complex material substrates. We argue that many physical substrates, even systems that in their natural state might not form usable or "good" reservoirs, can be configured into working reservoirs given some stimulation. To achieve this we apply techniques from evolution *in materio* whereby configuration is through evolved input-output signal mappings and targeted stimuli.

In preliminary experiments the combined model and configuration method is applied to carbon nanotube/polymer composites. The results show substrates can be configured and trained as reservoir computers of varying quality. It is shown that applying the reservoir model adds greater functionality and programmability to physical substrates, without sacrificing performance. Next, the weaknesses of the technique are addressed, with the creation of new high input-output hardware system and an alternative multi-substrate framework. Lastly, a substantial effort is put into characterising the *quality* of a substrate for reservoir computing, i.e its ability to realise many reservoirs. From this, a methodological framework is devised. Using the framework, radically different computing substrates are compared and assessed, something previously not possible. As a result, a new understanding of the relationships between substrate, tasks and properties is possible, outlining the way for future exploration and optimisation of new computing substrates.

Contents

List of Tables

List of Figures

Dedication

First, I want to thank my supervisors Susan, Martin and Julian. Their feedback and encouragement has been resolute throughout my time as a PhD student. The time and commitment they gave to me and continue to give to their students is inspirational.

I want to thank my family for being there throughout my unconventional route to academia and nudging me at those few critical moments when I needed it most.

Lastly, I want to thank my fiancé who joined me on this journey. She saw the gradual descent into madness, with late nights and weekends spent watching “evolution TV”, and the infamous “just five more minutes” turning into several hours. Without her patience, and help from many others throughout my time at York, this thesis would not exist.

Acknowledgements

Funding for this research was given through a Defence Science and Technology Laboratory (DSTL) PhD studentship.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. A large portion of the presented work has been published in co-authored papers, however all work presented is original. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Publications

Dale, M., 2015, October. Unconventional Reservoir Computers: Exploiting Materials to Perform Computation. In Eighth York Doctoral Symposium on Computer Science & Electronics (p. 69).

Dale, M., Miller, J.F., Stepney, S. and Trefzer, M.A., 2016, July. Evolving carbon nanotube reservoir computers. In International Conference on Unconventional Computation and Natural Computation (pp. 49-61). Springer, Cham.

Dale, M., Miller, J.F., Stepney, S. and Trefzer, M.A., 2016, Reservoir Computing: Evolution in materio's Missing Link. In Ninth York Doctoral Symposium on Computer Science & Electronics (p. 57).

Dale, M., Stepney, S., Miller, J.F. and Trefzer, M., 2016, December. Reservoir computing in materio: An evaluation of configuration through evolution. In Computational Intelligence (SSCI), 2016 IEEE Symposium Series on (pp. 1-8). IEEE.

Dale, M., Stepney, S., Miller, J.F. and Trefzer, M., 2017, May. Reservoir computing in materio: A computational framework for in materio computing. In Neural Networks (IJCNN), 2017 International Joint Conference on (pp. 2178-2185). IEEE.

Dale, M., Miller, J.F. and Stepney, S., 2017. Reservoir computing as a model for in-materio computing. In Advances in Unconventional Computing (pp. 533-571). Springer, Cham.

Dale, M., 2018, Neuroevolution of Hierarchical Reservoir Computers. In Proceedings of the Genetic an Evolutionary Computation Conference 2018 (GECCO '18). ACM.

Konkoli, Z., Nichele, S., Dale, M., and Stepney, S., 2018, Reservoir computing with computational matter. In Computational Matter (pp. 269-293). Springer.

Dale, M., Miller, J.F., Stepney, S. and Trefzer, M.A., 2018 (submitted) A Substrate-Independent Framework to Characterise Reservoir Computers. Proceedings of the Royal Society A.

Chapter 1

Introduction

Biological organisms vastly outperform classical/conventional computing paradigms in many respects, from possessing inherent fault-tolerance to constructing highly parallel machines. Much of this is achieved by exploiting physicality and by sharing and distributing computational effort throughout the spatial system. As such, they exploit physical interactions through feedback with the real world, utilising features such as their own morphology.

Many of these systems comprise relatively simple elements that emerge and coalesce into more complex, but robust, structural layers across different scales. Such grounding properties (and many more) have enabled these complex systems to thrive and evolve, adapting and co-evolving in real-time with their local ecosystem.

In modern science, many attempt to mimic the computational properties and behaviours of biological systems on conventional machines. It is argued here that such attempts can themselves be flawed. Many of these experiments attempt to imitate the performance of an embodied system in a non-embodied abstraction, in a process that requires some transformation to a symbolic representation. In essence, such transformations might detract from many of the physical aspects that make these natural systems so powerful.

The limitations of conventional computing paradigms are well defined; and we are rapidly approaching the limitations of current CMOS technology [115]. For technology to move forward, many of these limitations need to be solved, whether that be using the same classical paradigms or using alternative ones. For example, conventional computing is based on the manipulation of bits (0 or 1s) and boolean logic requiring formal mathematical definitions. However, to solve more complex tasks may require some intractable properties that cannot be formally defined.

The conventional von Neumann computing architecture, based on the stored-program computer concept, although expertly refined over decades, poses some fundamental inefficiencies. For example, classical computers require the transformation between high-level languages to low-level machine code, a process that requires layers of conversion through a compiler stack, making it computationally costly, slow, and highly susceptible to faults and errors.

These systems typically succumb to many issues in speed, from both an inability to deal with concurrent computations and the bottleneck created by the transfer of data between separate memory and processing entities. Because of these architectural weaknesses, other intertwined issues arise, such as an increase in power consumption, system size, and top-down design complexity.

1.1 Unconventional Computing

Unconventional computing tries to address some of the above limitations by attempting to provide alternative architectures and systems that typically exploit the underlying physics and many-scale interactions of the real world. Many forms of unconventional systems have been explored in recent years.

For example, quantum computing is one such system, where two-state quantum bits, typically described by electron or photon spin/polarisation, can be exploited to perform large numbers of parallel computations (i.e. $2^{\text{no. of qubits}}$). This is achieved through the principles of superposition, enabling a qubit to be in both states simultaneously (see [175] for a review).

Another example is reaction-diffusion computing [4], which performs computation through local chemical reactions and diffusion. By using chemical processes, the system can execute highly parallel computations, performed by the complex interactions of propagated waves of information caused by local disturbances. The result of such computations are stored, or encoded as the concentration profiles of the chemical reagents. Physarum polycephalum (slime mould) is currently under investigation as an excitable, reaction-diffusion medium that could form the basis of a programmable amorphous biological computer. The Physarum chip, a result of the PhyChip project [5], is described as consisting of networked protoplasmic tubes coated with conductive substances to create a living non-linear system. Experiments to-date with Physarum have already demonstrated an ability to approximate logic gates and derive combinational logic circuits [201].

Even earlier examples of unconventional systems can be found back in the mid twentieth century cybernetics movement in Pask's experiments with electrochemical assemblages in ferrous sulphate solutions [30, 154]. Pask attempted to evolve functional dendrite-like structures by passing current directly through immersed electrodes. To determine the growth of these structures one would simply define which electrodes to pass current between, with the resulting linkage conductance representing synaptic weights. In his experiments, Pask used manual selection to "evolve" linkages sensitive to perturbations caused by sound, or magnetic fields, to create a self-assembling "ear" that could be trained to discriminate between different frequencies.

Interdisciplinary research into unconventional computing has shown that many diverse dynamical systems can be exploited to perform computational tasks [3, 1, 2]. Exploiting computation directly at the substrate-level offers potential advantages over classical computing architectures, such as exploiting physical and material constraints that could offer solutions "for free", or at least computationally cheaper [178].

Extracting computation from these systems and physically *programming* them, however, can be very challenging. Sometimes this is further exacerbated by a desire for minimal abstraction, i.e., exploiting emergent physical phenomena directly, and wanting a level of programmability that enables the system to perform a variety of tasks.

Hybrid digital-analogue computers have the potential to fulfil these needs, where the digital system is trained to extract computation performed implicitly from an analogue substrate. Mills' Kirchhoff-Lukasiewicz Machines (KLM) [136], based on Rubel's computational model of analogue computation (the extended analogue computer [159]), are a prime example.

For analogue computers the program and architecture may be indistinguishable or inseparable, i.e., to program the machine requires a change in the machine architecture. However, by placing additional layers within the machine's architecture the amount of change required is reducible, for example, by adding a standardised reconfigurable "middle" layer.

In Mills' KLM device, a middle layer is implemented using Lukasiewicz logic arrays that perform piecewise linear functions on measured values from the material. To control the logic arrays a vector of bits referred to as the "overlay" [136] is used. This overlay, representing the semantic "program", is used to describe the reconfigurable layer that exploits the implicit material function. The computational functions being utilised are therefore a result of the material's configuration, typically achieved through the selection of inputs, outputs and control functions/signals.

Mills describes this as a paradigm of *analogy*, [137] where the computing device is not explicitly told to perform an operation and provide a readable output, but rather trained to exploit an implicit function that results from the material's configuration.

1.2 Configuring Matter to Compute

Conventional, classic computers are designed to be substrate-independent, where a symbolic "machine" is highly constrained into physical hardware.

Material/substrate-based computers are conceptual machines in which some computational process, or physical mechanism, maybe extracted from a behaviourally "rich" dynamical system. In essence, information processing can be exploited from what the substrate does naturally, for example, how the system reacts and dynamically adapts to some input stimulus [178]. Informally speaking, this can be viewed as "kicking" the dynamical system and observing its behaviour to some given stimulus, where the method of perturbation and observation may vary in type: electrical, mechanical, optical, etc. From this, we speculate as to whether some exploitable process, or *computation*, can be extracted and whether the system can be trained to consistently exploit this process.

Exploiting computation directly from materials offers many potential advantages over classical systems. As a paradigm, it potentially offers vast amounts of computational power by capitalising on the massive parallelism and natural constraints of these rich systems.

Recently, a similar computing paradigm to Mills' KLM – without the middle "readout" layer – has been applied to computational composites consisting of randomly dispersed carbon nanotubes. These semi-conducting substrates are shown to exhibit interesting computational properties across a variety of tasks using computer-evolved configurations [19]. Applying evolved control signals, the unknown electrical properties of these composites are perturbed and trained to produce physical solutions to computational tasks.

Training these composites is done through the evolutionary selection of input-output mappings and evolvable control stimuli, using a technique referred to as evolution *in materio* (EiM) [134]. The training method searches for material configurations that display interesting computational features related to a given task, in a vast space of configurations that is constrained only by physical laws.

To date, research based on the evolution *in materio* concept has showed much promise [19], however, many of the same issues identified earlier in exploratory work has yet to be resolved.

For example, the "analogy" programming method works well, but a middle layer providing some level of abstraction is still missing. This lack of abstraction leads to scaling problems in both hardware connectivity and task complexity. This limits its usefulness and applicability, and potentially reduces its adoption by the wider computing community.

Another fundamental issue is the non-trivial task of analysing what is being exploited, intrinsically, externally, and in terms of general computational and dynamical properties of the substrate. This is closely linked with how to determine if a substrate is suitably "rich" to solve computational tasks; and, if the substrate is suitable, what spectrum of tasks are appropriate.

At a basic level, a generic methodology for substrate characterisation is missing. At the moment, too much is expected from the evolution process: we cannot just give the algorithm Lead and expect it to turn it into Gold. The minimal criteria for evolution *in materio* to excel should be assessed on a substrate-by-substrate basis. Therefore, the substrate should be characterised prior to the evolutionary process and its potential determined.

1.3 Research Hypothesis and Thesis Overview

The research hypothesis is that a computational framework called *Reservoir Computing* (RC) can provide solutions to many of the present issues with evolution *in materio* (EiM), from providing a layer of abstraction to quantifying computational processes being exploited in the substrate. A full description of reservoir computing and how to apply it is given in Chapter 2.

Throughout the thesis, techniques from both EiM and RC are utilised, and where each falters the other is used to provide a solution. Reservoir Computing is not a given in the thesis, but is itself explored, and its definition expanded.

To test the hypothesis, several unconventional computing devices (based on carbon nanotube composites) are evolved *in materio* and trained *in silico*, creating a new hybrid computing framework, referred to here as *Reservoir Computing in materio* (RCiM).

First, the new hybrid computing framework is evaluated directly on machine learning tasks common to RC (Chapter 3). Then it is analysed in greater detail on different tasks (Chapter 4). Next, RCiM's combined low-level exploitation and abstraction is evaluated and compared to direct EiM (Chapter 5).

In order to take full advantage of the RCiM approach, a new extended hardware platform has been built. Details of its design, limitations and practical considerations when in-use are outlined in Chapter 6.

In Chapter 7, some limitations of reservoir computers are addressed. Here, ways to improve and extend the RC/RCiM framework are demonstrated with hierarchical reservoir structures, providing greater potential to tackle harder tasks, and perform better across multiple tasks.

Finally, in Chapter 8, a generic framework is defined to characterise any substrate for RC/RCiM, filling a significant gap both in evolution *in materio* and reservoir computing with physical systems. The new framework provides not only a way to judge the richness, or "quality" of a substrate but allows us: i) to better understand the computational processes being exploited; and ii) to determine and predict tasks appropriate for the substrate, in some cases without being assessed on tasks directly.

The novel contributions of this thesis are new frameworks to exploit and understand material computation as never before. With the combination of computer-controlled evolution and reservoir computing theory and abstraction leading to: greater performances and flexibility than previous work in evolution *in materio*; a new evolvable architecture to extend both the RC and RCiM framework to tackle tasks with greater complexity; and lastly, a generic framework to measure and characterise the quality of any substrate for reservoir computing. As a result, a significant step forward has been made in the design and understanding of physical reservoir computers.

Chapter 2

Context

2.1 Material Computation

Material computation is defined here as a computational process that occurs in a sufficiently complex dynamical system realised in the form of a physical material substrate.

This definition relies on the idea that matter contains physical information and that systems of collective matter can dynamically modulate and redistribute information to change state, therefore, implied here, to perform some form of computation. A simple example of this would be the physical process in which the state or phase of matter changes in relation to energy, which could (loosely) in some sense be viewed as a computational mechanism.

Stepney *et al.* [180] provide a more contextual definition of material computation as: "computation directly by physical and chemical processes of a complex substrate, with little or no abstraction to a virtual machine". From this definition we can categorise material computation as an analogue process that utilises physical constructs, the tangible medium itself, and meta-processing to do computation. This "physicality" may be defined and observed as the structural topology, characteristic behaviours and information processing associated with the many-scale interactions occurring in that system.

Material computation is still in its developmental or conceptual stage, with early experiments supporting, or working in tandem with, current digital technology to form hybrid, and potentially very powerful, machines. A configurable substrate can be used to transfer some of the computational burden from the digital system to the material [180].

As such, the material can endow the system with many of the properties and advantages of analogue systems, such as speed and concurrency, in a device where memory and processing are not separable. To achieve this, engineers attempt to exploit processes and behavioural phenomena that naturally occur, properties that are governed directly by the underlying physics and chemistry of the substrate.

Exploiting computation directly from materials offers many potential advantages over classical systems where the computation performed does not depend so much on the details of the materials used. As a paradigm, substrate computing potentially offers vast amounts of computational power by capitalising on the massive parallelism and natural constraints of these rich systems. Such properties are suggested to have the potential to provide solutions “for free”, or at least computationally cheaper, and provide a rich explorable state space, aligning computation to particular trajectories [178].

Much of the current interest in material computation is to abstract a model (or models) of computation from what the substrate does naturally. It has been proposed that the first step to producing a potential unified theory of material computation, i.e., a theory that better understands what computation is and how it occurs in materials, should take place in “primitive” (un-evolved) substrates, where the general principles are in plain sight [178]. After this, material computation could emerge with some supportive reasoning to a better understanding of computation in biological substrates.

2.1.1 The effects of criticality on complexity

It has been hypothesised that there is a critical state in which a system can exhibit maximal computational power, and therefore where maximal complexity can be acquired, labelled a region near (or at) the *edge of chaos* [103]. This concept may have a direct relationship to material computation, whereby a material can exhibit “richness” [134], and therefore be exploitable, by operating close to or within this region.

This *edge of chaos* represents the transitional border between ordered dynamics, where perturbations to the system quickly fade and the system settles, and chaotic behaviour, where perturbations significantly affect long-term stability and the system becomes unpredictable. This critical landscape can be observed by looking at the system’s trajectory in the phase space by monitoring the convergence towards or away from a steady state, and thus highlighting a system's sensitivity to initial conditions. Both behaviours are thought to be necessary to gain maximal complexity, using some ordered behaviour to maintain memory, and some chaotic behaviour to enable processing.

Langton [103] observed the effects and advantages of systems working in this transitional region, using cellular automata. At a critical point, Langton observed that a cellular automaton could optimally perform computations, imitating complex life-like behaviour. Earlier, Packard [152] observed another unique property: that genetic algorithms tend to evolve populations in these critical regions, suggesting that adaptability was therefore optimised close to the edge of chaos. Similar conclusions are proposed and demonstrated in neural networks, where vast computational power and capability in this region is described through network connectivity. Bertschinger and Natschläger [15] demonstrate these relationships in input-driven networks, by accurately determining the position of the critical line with respect to structural parameters.

It has also been suggested that *living* neural networks support the "criticality hypothesis". Beggs [13] discusses this notion by looking at how the power-law distribution of neuronal avalanche sizes (a cascade of bursts of activity) suggests operation near a critical point. Beggs further explains that the implications of these avalanche size distributions implies that information transmission, information storage, computational power and stability could all simultaneously optimise at the edge of chaos.

2.1.2 Configuration and structure

Conventional programs and algorithms represent idealised mathematical objects, irrespective of their underlying hardware. In a physical system, say a biological system, computation is embodied, and behaviour may not be completely captured by a closed mathematical model. As such, trying to program these embodied systems requires different techniques. The programming and manipulation of materials requires (to some extent) a complex understanding of the properties and interactions within that system. We therefore require either some convoluted top-down "programming" approach in the traditional sense, or an alternative mechanism, e.g. through training, learning or evolution. Whichever method is applied, this "program" would alter the details of system in a controlled manner, for example, using controlling fields that affect structure and dynamics.

Structure plays a pivotal role in the complexity, behaviour and programmability of a material. The structure may ultimately define the computation and potentially adhere exclusively to the task. In a practical sense, a material would ideally require some form of self-assembly, or self-organisation, to create bottom-up structures on the molecular scale. Traditional top-down design is not only impractical where exact arrangements are not possible, but also typically implies some form of uniformity, which may be detrimental when large varying dynamics are wanted.

The manipulation of silicon in traditional technology represents "what we do not want" in our exploitable substrates, i.e., a material that is given intentional limits to reduce the likelihood of "unwanted" interactions, a property argued here as detrimental to a device that requires rich exploitable behaviour.

2.1.3 Analogue computers

Material computation is principally a form of analogue computation where no symbolic representation is created and the physics of the continuous system affect computability. Known variations of analogue computers date back as far as the second century BCE, where mechanical analogue computers were used for astronomical purposes [129]. Most analogue computers, then and now, are typically devoted to one task, e.g., the ancient Greek Antikythera mechanism used for astronomical calculations, and the Torpedo Data Computer used for torpedo fire-control on a World War Two submarine.

General-purpose, or *universal* machines (although universality in computation is hotly debated [6, 7]), are desirable in both analogue and digital domains, i.e., one machine that can implement all others. Universal machines are useful in highlighting the limits of domain-specific computation. For example, what is not computable on a universal machine is not computable on any other computer in that domain. The seeds of universal analogue computers arose in the nineteenth century from Lord Kelvin's differential analyser, which was later built by Bush to solve differential equations [24, 129].

Shannon devised a mathematical theory of Bush's mechanical computer, which led to a formalisation of the general purpose analogue computer (GPAC) [168]. The computer was designed to solve initial-value problems for ordinary differential equations (ODEs). Once the invention of the operational amplifier arrived, mechanical analogue computers were quickly replaced by faster and more cost-effective electronic versions. These computers operated on continuous variables using a range of linear and non-linear elements such as adders, multipliers, integrators and function generators.

Over time, electronic analogue computers became largely overshadowed by their digital counterparts. After a few decades of incredible gains in digital computers and reduced interest, education and understanding of the advantages of analogue computation, it diminished somewhat as a viable field to investigate. However, analogue computing never completely disappeared; in many cases it just took on a different form. For example, hybrid systems became popular in critical real-time applications as they still vastly outperformed digital systems, e.g., in areas such as the US intercontinental ballistic missile program in the 1950s [129].

In the 1990s another general-purpose mathematical model was proposed by Rubel, known as the Extended Analogue Computer (EAC) [159]. Rubel's conceptual computer (the EAC) tackles the limitations of Shannon's GPAC by extending it beyond ODE problems, allowing it to compute differential algebraic functions of any finite number of real variables (not just time). Principally, an EAC integrates a GPAC and a partial differential equation (PDE) solver. Mills went on to create a practical EAC, despite Rubel's reservations on its electronic tangibility, and named it the Kirchhoff-Lukasiewicz Machine (KLM) [128]. Using a form of multi-valued logic, known as Luckasiewicz logic, Mills could overcome the perceived difficulties by approximating GPACs and therefore solve ODEs in the KLM system (see [136]).

KLMs are composed of three computational components: i) the partial differential equation solver, implemented as conductive sheets; ii) the reconfigurable Lukasiewicz logic arrays (LLA), representing piecewise linear function generators; and iii) wire junctions that perform addition and subtraction using Kirchhoff's current laws [136]. The machine is configured, or "programmed" in some sense similar to a digital computer where a vector of bits (the overlay) reconfigures all components.

Configuration of a KLM is typically a manual process, although the use of genetic algorithms (GAs) to create overlays is briefly mentioned in [136, 137]. In earlier work, Mills' research students evolved overlays that produced "encouraging" results. Later, Harding successfully evolved configurations to symbolic equations, "bridging the semantic gap between symbolic mathematical functions and the EAC architecture" [137]. This feat was achieved by evolving a function of the voltage at a location (x,y) on the conductive foam sheet.

2.2 Evolution *in materio*

Evolution *in materio* (EiM) is a term coined by Miller & Downing [134] to refer to the means by which a physical system, a complex material, could be manipulated by computer controlled evolution (CCE) to perform useful computation.

The idea of using unconstrained artificial evolution as a search method in physical media is deeply rooted in the field of Evolvable Hardware (EH) [68, 71]. Most evolved configurations in EH lead to digital products or components later embedded into physical artefacts. Examples include: evolving simulated models and optimisation programs; designing a physical system that can be manufactured after evolution, like antennas [116], robots [111, 160] or chemical systems made of oil droplets [70].

Natural evolution is a fully *embodied* (intrinsic) process where physical systems die, reproduce/self-replicate, co-evolve with the environment and other organisms, becoming many physical instantiations and tweaked copies of the thing being evolved. Artificial evolution represents varying layers of embodiment. Eiben *et al* [53] propose three forms, with most evolvable systems coming under the first two: digital and physical instantiations as a product of evolution in simulation. The third is fully embodied evolution, where evolution is occurring within the product being created.

Miller argues that evolution *in materio* sits between full embodiment and the realised evolved EH solutions described above [135]. In this form, physical artefacts are configured (or conceptually created) during the process and assessed/controlled by simulated Darwinian evolution. This can also be seen in some EH systems, but, typically such systems are limited to constrained silicon hardware, e.g., electronic circuits evolved on Field Programmable Gate Arrays (FPGAs) [79].

EiM relies on a hybrid analogue/digital architecture where the evolutionary process (run on a digital computer) controls the writing/reading of physical signals to/from an analogue material. The directed search tries to exploit the dynamics of the material by evaluating the performance of individual test configurations. Physical realisations are therefore embodied in the search process but evaluated externally. A system that operates in this manner is theoretically very powerful, allowing the manipulation of physical properties which are hitherto unknown.

An early example of EiM can be found in Thompson's work with FPGAs [187]. Thompson evolved a frequency discriminator by allowing evolution to reconfigure circuit elements on the FPGA. He discovered that evolution had exploited subtle electrical variations in the underlying material to form a solution. This was made evident when evolved configurations no longer solved the problem when moved around the FPGA and when areas not directly connected were nevertheless found to contribute to the overall operation. Thompson's seminal work led to an explosion of interest in intrinsic evolution [71, 135].

An advantage – but which should also be viewed with caution – of the EiM methodology is that the training process does not require a full understanding of the computational mechanisms that it exploits, treating the material as an encapsulated *black-box*. These systems are also *open* systems, where energy can be lost to and gained from the environment. Therefore, some exploited mechanisms may be non-trivial to analyse, requiring analysis at a holistic level across multiple scales, e.g. substrate-level, system-level, and environmental.

The training procedure widely used in EiM takes the form of evolving a set of signals and their connection locations on an electrode array interfacing the computational material. Evolution is not the only possible training procedure: there are many ways to manipulate and interact with a physical system. Other training algorithms linked to EiM include particle swarm optimisation [198] and global and local search [69]. The general aim of all of these is to find, or optimise, an input-output mapping that carries out a desired computational mapping. The rationale is that physical systems contain enormous complexity, and that evolution may exhibit the most efficient method to discover and exploit these physical properties.

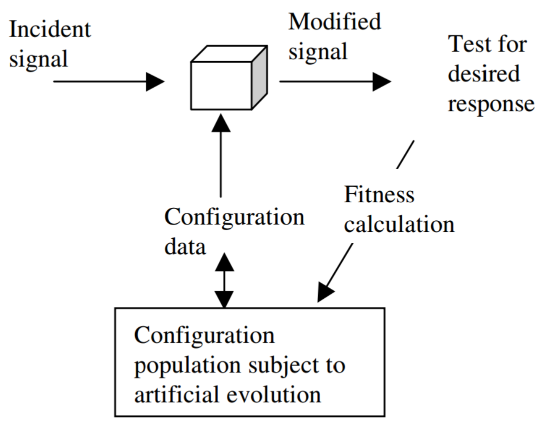

Incident signal
Modified signal
Test for desired response
Fitness calculation
Configuration data
Configuration population subject to artificial evolution

Figure 2.1: Configurable Analogue Processor (CAP): material is reconfigured to solve some computational task through applied input signals [134]

2.2.1 Computation in Liquid Crystal

Miller & Downing [134] discuss several interesting materials that could possess the desired characteristics needed for both computation and evolution. These include: liquid crystal, conducting and electro-active polymers, voltage controlled colloids, nanoparticle suspensions, and irradiated or damaged semiconductors [134]. To exploit these materials, a configurable analogue processor (CAP) (see Fig. 2.1) was proposed to alter the material's function through configuration parameters (discrete signals).

Harding & Miller [73] adopted liquid crystal as a material and constructed a bespoke platform around it to test the concept, and solve multiple computational problems. Liquid Crystal (LC) has a number of advantages for readily applying EiM. LC contains several key features including: has wide availability, is addressable using digital voltages, exhibits emergent behaviour, has a unique mesomorphic structure between ordered and disordered, and can relax to an initial base state.

Harding & Miller's platform houses a liquid crystal display (LCD) and an array of dynamically selectable input/output connections to both the LCD and external measurement devices. Over the course of their investigations, the LC system was applied to three separate tasks: tone discrimination [73], creating logic gates [75], and a real-time robot controller [74].

They demonstrated liquid crystal as an efficient evolvable material where relatively small numbers of generations could produce effective solutions. They also found that a

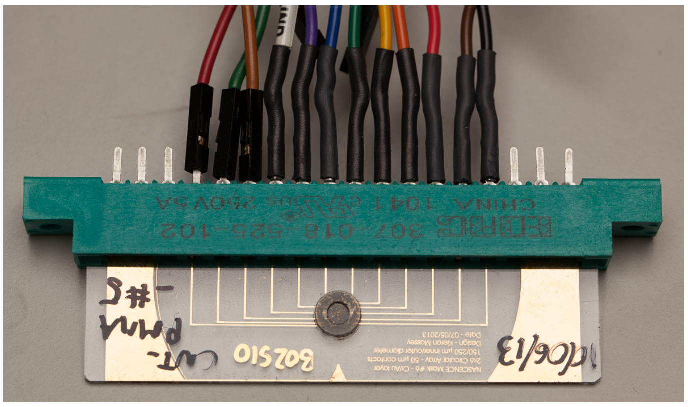


Figure 2.2: Micro-electrode array used in the NASCENCE project to contain, stimulate and record activity in a carbon nanotube-based substrate. Computer controlled evolution is used to select active electrodes and mode (i.e. record or stimulate)

rich substrate of liquid crystal supplied many more exploitable properties compared to conventional silicon hardware such as Thompson's FPGA. This increased the diversity of solutions, thus increasing its evolvability.

This embryonic work demonstrated the advantages of emergent design by configuring intractable characteristic properties with no knowledge of their existence. But, the experiments also raised other fundamental questions on applying the evolution *in materio* technique, for example, the length of time needed to "program" materials, what is actually doing the computation, and how reproducible are solutions when environmental conditions change. Many of these questions and more are discussed in [135].

2.2.2 NASCENCE project: Carbon nanotube substrates

As part of the European FP7-ICT research project NASCENCE[1] (NAnoSCale Engineering for Novel Computation using Evolution) further materials were considered, along with a new bespoke hardware platform [18]. The hardware platform, known as the Mecobo board [122], forms another hybrid hardware architecture to integrate digital computers with experimental materials. The system interfaces with materials placed on micro-electrode arrays (MEA) (Fig. 2.2) using a similar premise to Harding & Miller's liquid crystal system, the CAP.

[1]NASCENCE homepage: nascence.no

The primary substrate used in the NASCENCE project consisted of Single-Walled Carbon Nanotubes (SWCNT) mixed with either a polymer or liquid crystal. The polymer/LC mixtures disperse the nanotubes into random static networks, forming varying connection topologies and conductive pathways, possibly forming something akin to a random electrical circuit. Carbon nanotubes are used as they can exhibit either metallic or semi-conducting behaviour and contain other unique properties (e.g. ballistic conduction, thermal conductivity, self-assembly via van der Waals forces), whilst the mixing material is believed to create isolating regions, forming an insulator between elements and creating network structure.

Another substrate consisting of randomly-dispersed gold nanoparticles was also investigated, showing very promising non-linear characteristics, but required cryogenic temperatures of less than a few Kelvin to function [17].

Many investigations within the project demonstrated the capabilities of SWCNT/ polymer mixtures, in particular poly-methyl-methacrylate (PMMA) and poly-butyl-methacrylate (PBMA), as a potentially rich, evolvable and computationally generic substrate. Computational tasks addressed include: solving classification and optimisation problems such as frequency classification [139]; classifying various data instances [35, 141]; solving small numbers of cities instances of the travelling salesman problem (TSP) [34]; and applied to the (NP-hard) bin packing problem [140]. These early results demonstrate the potential of the methodology, but, in some respects it still lacks competitive results and still exhibits some of the issues raised in Harding & Miller's work with LC [135].

PBMA composites show greater stability than PMMA composites. The electrical percolation threshold of PBMA is claimed to occur around a SWCNT concentration of 1% (w.r.t. polymer weight), forming a useful mixture of short and long-conductive pathways. Adding further nanotubes to the mixture is claimed to provide a negligible computational advantage as a suitable network appears to already exist. However, higher concentrations do demonstrate more non-linear current–voltage (I–V) behaviour in comparison. At less than 1%, the nanotube networks become fairly sparse, and are claimed to have reduced computational performance (and potentially more linear properties) [130]. The PMMA polymer was investigated towards the beginning of the NASCENCE project (see [100]) and is reported to have a percolation threshold between 0.17 and 0.70% in the literature [130]. However, a direct comparison is difficult to make between the two polymers as they come under different polymer groups, i.e. contain different chain lengths.

More recent investigations into SWCNT/liquid crystal mixtures have shown promise [131, 198], for example, the non-linear I–V behaviour appears to be more prominent. It has also been demonstrated that conductivity and orientation can be changed by an in-plane electric field. But LC has been shown to experience a longer configuration time, in terms of LC molecule and SWCNT alignment, due to LC molecules being smaller than SWCNT ribbons, and associated relaxation times. Other issues include: long-term stability and reconfigurability, and the exact role of the liquid crystal in nanotube alignment [199].

2.2.3 Summary

In general, evolution *in materio* is still in its infancy. New engineering technologies, adaptations in the search method (or fitness criteria), changes in hardware (number of electrodes, different pitch sizes, etc.) and new materials could all have a significant effect on the field. The key components to the success of current EiM substrates lay in improvements to the fabrication of materials and the interfacing system used for stimulation and observation.

So far, early work has highlighted only one of many means of "programming" a material via evolution (i.e., through discrete voltage inputs). Other controlling fields and mechanisms (magnetic, thermal, optical, etc.), or even a combination, could be utilised to manipulate and configure different materials, further increasing the distinction between configuration and input signals. Many potential systems that possess such controlling mechanisms have emerged recently (see section 2.3.8), but none have yet adopted the evolution *in materio* concept.

2.3 Reservoir Computing

Reservoir Computing is the unification of three individually conceived methods for creating and training artificial recurrent neural networks (RNN): Echo State Networks (ESN) [86], Liquid State Machines (LSM) [127] and the Backpropagation-Decorrelation

(BPDC) on-line learning rule [176].

A typical RNN model consists of a system of three layers: an input layer, a hidden layer (the core network), and an output layer (see Fig. 2.3). The hidden layer contains processing elements (neurons) that are interconnected through weighted synapses (connection weights). The input and output layers are connected to the hidden layer, again through weighted synapses. Variations on the types of connectivity, e.g. feedback from the output to hidden layer or input layer to output layer, depends on the task and method.

When driven by an input, neurons activate, propagating information through the network to other neurons through varying connection strengths. The presence of recurrent connections can produce self-sustained activations, preserving a dynamic memory in the network's internal state.

Networks such as this have been shown to be theoretically very powerful and can be both Turing equivalent [97] and good universal approximators of dynamical systems [60]. However, making the most of RNNs comes at a price, as they suffer from many training difficulties, such as the computational expense of updating large networks, bifurcation points, and sometimes falling into inescapable local optima when using gradient-descent.

Reservoir Computing offers an alternative training technique. It reduces the computational cost and removes the problem of degenerative gradient information that leads to poor convergence in RNNs. But, the reservoir computing framework also goes beyond traditional neural networks. Its training simplicity, black-box nature, and exploitation of widespread dynamical characteristics allow it to encompass many more dynamical systems than just RNNs.

There are many "flavours" of reservoir, originating from two separate research fields of machine learning and computational neuroscience. The first focusses on training dynamical systems for temporal learning tasks using artificial recurrent neural networks. The second aims at realistically modelling the computational properties of neural microcircuits. A brief summary of the two main branches of RC are given below. For more types and variations see [119].

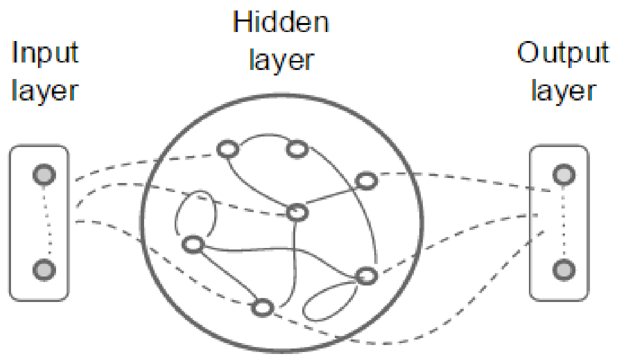


Figure 2.3: Example three-layer recurrent neural network.

2.3.1 Echo State Network

The *Echo State Network* (ESN) is a discrete-time neural network constructed from a sparse, random collection of analogue neurons. The typical neuron model employed uses the sum of its weighted inputs, applied to a sigmoid function (generally a hyperbolic-tangent), to give the neuron state $x(n)$ at time n. The state activations $x(n)$ of these neurons are termed as echo states [86], i.e. echoes of the input history. To propagate and hold this history the network requires the *echo state property*, or more generally speaking, a fading memory. The property itself is provided by the characteristic dynamics of the system.

An example ESN is given in Fig. 2.4. This consists of a random, static recurrent network of neurons. The input $u(n)$ is fed to the reservoir network W via the input layer weight matrix W^{in}. An extra input W_{bias} (labelled "1" in the figure) is provided to bias the activation functions.

Each node/neuron in the hidden layer network (the reservoir) possesses a one-to-one weighted connection to all nodes via the internal weight matrix W; only a few weights are non-zero, making the network sparse. When driven by the input, input states, network states and echoes of each are captured in the reservoir network and observed as states $x(n)$.

The output layer represented by the weight matrix W^{out} is trained using the reservoir states $x(n)$ and target signal $y^{target}(n)$ to minimise the error E between $y(n)$ and $y^{target}(n)$.

For ESNs, different scaling parameters, and in particular the *spectral radius* $\rho(.)$, influence the characteristic dynamics of the system. These parameters fundamentally

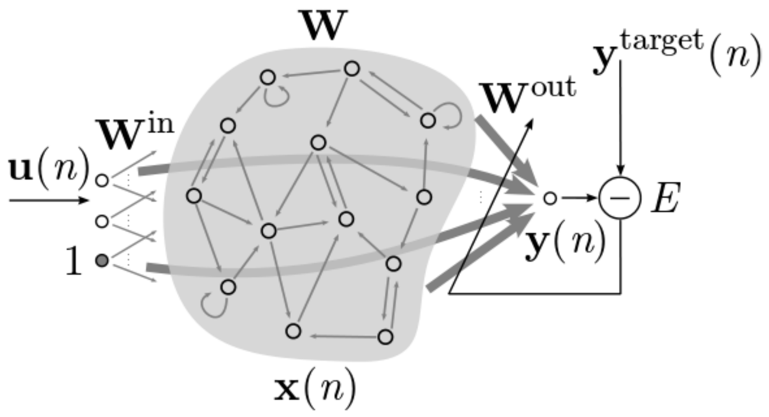


Figure 2.4: Echo State Network taken from [117]

alter and control the amount of memory and non-linearity present in the system. The $\rho(.)$ parameter is used to scale the weight matrix W so that the largest absolute eigenvalue generally satisfies $\rho(W) < 1$. Within this region the echo state property is said to be assured. However, this is not always necessary [119].

A variant of the ESN model is the leaky-integrator neuron model, using a neuron that possesses some form of memory of previous activations. These neurons contain a leaking rate, or decay parameter α, which can control the speed of the reservoirs update dynamics. The state update equation for leaky-integrator neurons is expressed as:

$$x(n) = (1-\alpha)x(n-1) + \alpha f(W_{in}u(n) + Wx(n-1)), \tag{2.1}$$

where f is the function of the neuron, typically a *tanh* sigmoid function.

As Jaeger [86] describes, each leaky-integrator neuron acts like a digital low-pass filter enabling a discrete network to approximate the dynamics of a continuous network (variations and uses can be seen in [86, 117, 119]). Dynamical systems have a natural time-scale; understanding the time-scale on which the input is changing compared to the time-scale of the system dynamics can be difficult. The leaky parameter α helps control and mediate any differences in input time-scales.

2.3.2 Liquid State Machine

The *Liquid State Machine* (LSM) model arose as a method for defining the computational properties and power of neural microcircuits, “an alternative to Turing and attractor-based models in dynamical systems" [126]. The LSM model represents a competitive model for describing computations in biological networks of neurons. The LSM attempts to model cortical micro-columns in the neocortex, structured in cortical layers of randomly created spiking neurons based on a spatial embedding.

Networks based on LSM use continuous streams of data (spike trains) to achieve real-time computations. Maass [126] argues that classical models cannot handle real-time data streams based on spike trains. Unlike ESNs, LSMs are generally more adaptive systems, supporting additional advanced readout features such as parallel perceptrons [127], although, in many cases, a linear readout is preferred.

Investigations using Liquid State Machines has highlighted the potential for more abstract applications, for example, pattern recognition using a physical medium (water) [55], and imitating LSMs in *E. Coli* [94].

2.3.3 Neuroscience

The conjecture that a simple linear classifier can extract complex computation from distributed neuronal activity is clearly pertinent to physical processes that occur within the brain. The brain encompasses a mass of highly complex architectures with specialised regions undergoing different tasks. Ultimately, one might perhaps ask: is RC a suitable metaphor for brain-like computing, or is it just another way of utilising neural networks and making them more trainable? Furthermore, if we think about regions of the brain acting as reservoirs from an evolutionary standpoint, how do these specialised parts evolve if they are not simply learned?

It has been shown that regions such as the primary visual and auditory cortex can operate in a similar fashion to a reservoir when an additional readout is applied. Nikolić *et al* [148] use the primary visual cortex of a cat to test whether a simple linear classier can extract information related to visual stimuli. Under anaesthesia, probes are placed into multiple architectural levels within the cortex to simultaneously record different neuronal activity. The visual stimuli used come in the form of different alphabetical letters to induce spike train activity.

It was discovered that: the system's memory for the past stimulation is not necessarily erased by the presentation of a new stimulus, but it is instead possible to extract information about multiple stimuli simultaneously. This implies that multiple sources of information, including that of previous states (i.e. memory) are superimposed upon the network. This reportedly allows XOR classification on the input through linear classifier extraction, directly showing that this region by itself is performing non-linear computation.

Klampfl et al. [98] show similar behaviour within the primary auditory cortex in response to tone sequences. Similar mechanisms have also been identified as a possible process used by mammalian brains in speech recognition [50].

2.3.4 Reservoirs and kernels

A reservoir can be interpreted as possessing kernel-like properties. A *kernel* acts as a pre-processor, embedding input data into a vector space known as a *feature* space. It is understood in many statistical machine learning methods that combining this feature space with a simple linear discriminant algorithm can enable the learning of complex non-linear functions.

In kernel methods this is achieved by projecting the input space $u(n)$ into a high-dimensional (possibly infinite-dimensional) feature space $x(n)$ *without* paying the price of its explicit computation, referred to as the *kernel trick* [162]. A kernel can therefore be expressed as the expansion function $x(u(n))$. However, there are two significant differences between reservoirs and kernels: reservoirs ***do*** explicitly compute the input transformation, i.e. do not possess the kernel trick; and kernels are typically ill-equipped to handle temporal signals. To tackle temporal tasks the learned function $x(.)$ requires some form of memory of previous inputs. Reservoirs solve this by utilising the network's recurrent topology, which creates memory by retaining previous state activations. The final expansion function of the reservoir can therefore be represented as $x(n) = x(x(n-1), u(n))$.

Reservoirs use this pre-processing technique to map temporal features of the input into a spatially defined feature map (the network). The desired features can then be extracted, or combined, in a linear fashion to create the output $y(n)$:

$$y(n) = W^{out} x[u(n)] \tag{2.2}$$

where $W^{out} \in \mathbb{R}^{N_y \times N_x}$, and N_y, N_x are the number of output nodes and internal nodes respectively.

This enables reservoirs to tackle many temporal and dynamic real-world tasks not possible using simple non-temporal kernels. Eqn. (2.2) also implies the system contains a clear separation between the reservoir and the linear readout, separating the training procedure from the hidden layer, i.e. only W^{out} is trained. As such, the kernel representation offers a much faster and better converging mechanism compared to other RNN models, as it does not suffer from vanishing gradients. This representation also classes reservoirs as powerful adaptive filters. For more information on kernels see [170].

2.3.5 Reservoirs and criticality

To design an optimal reservoir one should find a good trade-off between: (i) the transformation of the input that optimally boosts the linear classifiers capability, later referred to as the "*quality*" of the kernel; and (ii) a sufficiently-long (fading) memory based on the input history. These two properties often conflict: to obtain a useful memory requires ordered behaviour and a rich transformation requires dynamic behaviour. Legenstein & Maass [105] have shown that optimal reservoirs tend to experience the best trade-offs at a critical point near the "edge of chaos".

Dynamic networks are said to exhibit emergent criticality and self-organising properties [181]. These systems are characterised by motion in the phase space described as *trajectories* or state transitions. A trajectory may converge towards (i.e., be attracted to) a stable or unstable steady state; an attractor. Attractors vary from *point* – a point in the state space that attracts trajectories into its basin – to *strange* and chaotic – attracting trajectories, but inside diverge exponentially. These systems are very robust, small perturbations in the trajectory will tend to converge towards the same attractor. But, both external inputs and parameter changes in the system can drastically alter the shape of the phase space. Such changes may perturb a trajectory, moving or "clamping" the system between different attractor basins. As a result, clamping may even create, remove or change existing attractors and thus alter the initial phase space created by the natural dynamics of the system [179]. The overall dynamics of the system can therefore, to some extent, be controlled.

Early measurements of dynamics in reservoir-like systems can be found in [15] where, using a similar technique to Derrida & Pomeau [49], dynamic behaviour is measured using the Hamming distances between two output states. By observing a growth in the distance between states it can be determined that chaotic behaviour is present; a decrease in distance would indicate more ordered behaviour. This concept is similar to computing the characteristic Lyapunov exponent for a dynamical system, with both measures analysing the sensitivity to differences in initial conditions.

Bertschinger & Natschläger [15] highlight two fundamental properties required for these networks: a *fading memory* and a "network mediated" separation. A fading memory is indicative of an ordered phase (memory) with some dynamics (fading). The same property is found in both Liquid State Machines and Echo State Networks, referred to as the "echo state property" in the latter. This allows the readout function to use information from recent inputs and derive functions of those inputs from the network state. Network mediated separation is deemed fundamentally important for input time-series networks, with similarities to the separation property in LSMs (see [127]). The property requires (ideally a large) diversity in network states that is the result of differences in inputs alone, allowing characteristic features to be identified in the input, and that any changes in state should not directly be a result of chaotic dynamics which could produce the same phenomena. As such, this property enables a readout function to respond effectively to any variation in the inputs.

Legenstein & Maass [106] propose two critical elements that characterise the computational capabilities of a complex dynamical system (cortical neural microcircuits in this case). These new measures, or properties, are proposed because Lyapunov exponents are only useful for analysing a system's dynamics, and are not necessarily helpful in predicting good parameter regions that create high computational performance. These two measures are the *kernel-quality* and the *generalisation-capability*. The kernel-quality refers to the linear separation property found in kernels (section 2.3.4). An empirical measure of this property is achieved by examining the complexity of functions that can be carried out on the inputs that boost the classification power of a subsequent linear layer. The generalisation-capability quantifies a system's capability to generalise any learned behaviour to a new input.

Boedecker *et al.* [16] extend these ideas to ESNs, and create a general framework for direct and localised measurements for each neuron. Boedecker *et al.* give measurements indicating the memory of each neuron and the transfer of information between each neuron. This work highlights some interesting and relevant points for all systems: a network does not necessarily need to be at the edge of chaos to do computation, the measured region where a system is at the edge of chaos is not universal for all tasks, and a critical state may maximise computational capabilities, but, such criticality may also be unnecessary or detrimental to certain tasks.

2.3.6 Training reservoirs

The two methods for training the reservoir outputs, i.e. linear readouts, are: "off-line" *batch-mode* training, using simple linear or ridge regression techniques (done once all reservoir states are collected into matrix X for training length T), and "on-line" training, often gradient descent-based, Recursive Least Squares algorithm (a useful extensive investigation of RLS-type training is shown in [101]).

Off-line training

Reservoirs are traditionally trained in a supervised manner where the temporal input $u(n)$ and coupled target output $y^{target}(n)$ are provided. Given a desired output the system can learn input-output behaviour by minimising the error (Eqn. (2.3)) between system output and desired output:

$$E(y, y^{target}) = \sqrt{\frac{1}{T}\frac{\sum_{t=1}^{T}(y(n) - y^{target}(n))^2}{\sigma^2(y^{target}(n))}} \tag{2.3}$$

To evaluate if the learned behaviour generalises accordingly, new input data is tested and the error between the two are again compared.

The general update state equations for reservoir systems are defined in Eqns. (2.4) and (2.5) for, discrete time $n = 1, \ldots, T$, internal state $x(n)$ and output $y(n)$:

$$x(n) = f\left(W^{in}u(n) + Wx(n-1) + W^{fb}y(n-1) + W^{bias}\right) \tag{2.4}$$

$$y(n) = f^{out}\left(W^{out}[x(n); u(n)]\right) \tag{2.5}$$

The function f is commonly represented by a sigmoid, typically a hyperbolic tangent. In echo state networks this represents a basic *tanh* neuron, but varies depending on the application. In other networks, f can be designed to form linear nodes, threshold logic gates or spiking neurons. In regards to the output $y(n)$, f^{out} may also be a non-linear function (sigmoid) but tends to be the identity in most cases.

Reservoir computers are conceptually viewed as recurrent networks incorporating the three-layered topology of N_x hidden nodes (neurons), N_u inputs and N_y outputs. The weight matrices – input weights $W^{in} \in \mathbb{R}^{N_u \times N_x}$, reservoir/hidden layer weights $W \in \mathbb{R}^{N_x \times N_x}$ and feedback weights (from the output to the reservoir, if needed) $W^{fb} \in \mathbb{R}^{N_x \times N_y}$ – describe the network and are all drawn randomly from a uniform distribution at creation and remain static. The output weight matrix $W^{out} \in \mathbb{R}^{N_y \times (N_x + N_u)}$ is the only thing to change and includes weights for the inputs as they act as additional states; hence the concatenation of $[x(n); u(n)]$.

Typically, the W matrix forms a sparse network by setting many of the weights to zero, the other W^{in} and W^{fb} matrices can either be dense or sparse. Additional scaling parameters might also be applied to the matrices to govern properties such as non-linearity, stability and global dynamics.

The bias W^{bias} can be used to counteract training issues and large weights by adding noise, acting as a *regularisation* parameter, or to push the *tanh* neuron to a particular state, creating a smoothing effect.

Applying feedback W^{fb} can be useful, or detrimental, to certain tasks. Some tasks might not be learnt to a reasonable degree without feedback, or, certain systems may require dynamics beyond what is supplied by the driven input to construct a suitable output. Adding feedback comes with its own risks: feedback will ultimately change the stability of the system and requires adaptations in the training procedure. It is often advised to use feedback only when necessary. For more information see [117].

The *off-line* technique is completed in one training cycle T after the system has computed all states for the given inputs. It provides a very fast training mechanism as it essentially computes a linear model given by the known output Y, collected reservoir states X and desired output Y^{target}:

$$Y = W^{out} X \tag{2.6}$$

The collected state matrix $X \in \mathbb{R}^{N_x x T}$ is created when the input $u(n)$ is run through the reservoir states $x(n)$. To avoid initial transients created by an initial zero state $x(0)$, a section at the beginning of the training sequence is discarded in the state matrix X. Essentially, the system goes through a “warming-up” process where states bounce around rather than returning to the equilibrium output, i.e. the system is too chaotic to retrieve any useful information about the input.

Given Eqn. (2.6), we can find the optimal weights that minimise the error between $y(n)$ and $y^{target}(n)$ by solving the overdetermined system:

$$Y^{target} = W^{out}X \tag{2.7}$$

Equation (2.7) can be solved for W^{out} using linear regression. The simplest method is to use Ordinary Least-Squares (Eqn. (2.8)), but typically this method succumbs to stability issues when inverting (XX^T).

$$W^{out} = Y^{target}X^T(XX^T)^{-1} \tag{2.8}$$

Lukoševičius [117] recommends using either ridge regression (regression with Tikhonov regularisation β) (Eqn. (2.9)) or the Moore-Penrose pseudo-inverse (Eqn. (2.10)). Ridge regression is a stable and effective solution and is generally advised. Adding a regularisation parameter counteracts the problems of producing very large output weights, which often indicates very sensitive and unstable solutions. Ridge regression with Tikhonov regularisation is given as:

$$W^{out} = Y^{target}X^T(XX^T + \beta I)^{-1} \tag{2.9}$$

where I is the identity matrix and β the regularisation parameter.

Setting $\beta = 0$ gives the same method for solving linear regression in Eqn. (2.8). It is therefore recommended to use a logarithmic scale for selecting β where it never reaches zero [119].

The pseudo-inverse is applied in some cases typically because of it is straightforward to implement in certain programming environments (e.g. MATLAB). However this comes at a price. The pseudo-inverse method is computationally expensive for large matrices of X and typically overdetermined. However, in most cases the network is made up of relatively small matrices and over-fitting depends on the difficulty of the task. The pseudo-inverse training equation is given as:

$$W^{out} = Y^{target}X^{+} \tag{2.10}$$

where X^+ represents the pseudo-inverse function on X.

On-line training

Some tasks require an on-line training method that adapts with time, minimising the error at each time step. This implicitly turns W^{out} into an adaptive linear combiner. The Recursive Least Squares (RLS) algorithm (Eqn. (2.11)) is more commonly applied as it overcomes the severely impaired convergence performance of the Least Means Square (LMS) algorithm [88].

$$E(y, y^{target}, n) = \frac{1}{N_y} \sum_{i=1}^{N_y} \sum_{j=1}^{n} \lambda^{n-j} (y_i(j) - y_i^{target}(j))^2 \tag{2.11}$$

Using RLS comes at a cost: the number of weights is quadratic rather than linear, and it can still be numerically unstable in some cases. Other powerful on-line methods may be useful to a practitioner, particularly in the presence of feedback connections, such as Backpropagation-Decorrelation (BPDC) [176].

The RLS training procedure is described here, derived from [44]. First, set the error forgetting parameter λ close to but less than one; the forgetting parameter controls the contribution of previous samples. Next, initialise the autocorrelation matrix $\rho(0) = I/\delta$, with δ being a small constant and I the identity matrix. At each time step compute the output weights using the following steps:

Step 1: Calculate the reservoir state $x(n)$ and output signal $y(n)$ for input $u(n)$.

Step 2: Calculate the error between target output $y^{target}(n)$ and system output given previous output weights:

$$e(n) = y^{target}(n) - W^{out}(n-1)x(n) \tag{2.12}$$

Step 3: Update the gain vector $K(n)$:

$$K(n) = \frac{\rho(n-1)x(n)}{\lambda + x^T(n)\rho(n-1)x(n)} \tag{2.13}$$

Step 4: Update the autocorrelation matrix $\rho(n)$:

$$\rho(n) = \frac{1}{\lambda}\left(\rho(n-1) - K(n)x^T(n)\rho(n-1)\right) \tag{2.14}$$

Step 5: Assign new output weights $W^{out}(n)$ using (2.12) and (2.13):

$$W^{out}(n) = W^{out}(n-1) + K(n)e(n) \tag{2.15}$$

For more readout training methods including feedback training (such as FORCE training), supervised, unsupervised, reinforcement learning, etc., consult the review [119] and practical aid [117].

2.3.7 Adaptation and pre-training

Adaptive reservoirs, ones that change weights or configuration, are inspired by natural adaptation in biological neurons. These adaptive processes are the result of persistent changes in a neuron's electrical properties, governed by unsupervised *local* adaptation rules often referred to as *Intrinsic Plasticity* (IP). These rules represent a homeostatic mechanism in which neurons self-modify their intrinsic activity (i.e. excitability). Using such learning rules has been shown to increase robustness and performance when pre-training reservoirs [164, 177]. For an overview of past trends including both local and global adaptation schemes, see [118, 119].

In classical RC, reservoirs are generated randomly, hence the performance of each reservoir varies on creation. Reservoir computing boasts its training performance on the separation between the reservoir and readout. The readout training, at its core, is quite inexpensive, allowing other forms of reservoir *pre-training*, i.e. generating reservoirs deterministically for each task. Even a crude search such as selecting a reservoir which produces the smallest error from a pool of randomly-created reservoirs highlights the advantages of pre-training.

Evolutionary algorithms are one potential search strategy for pre-training. Investigations using evolutionary optimisation are well documented. Many strategies have been attempted, including evolving topologies (i.e. network size), weight matrices (such as W^{in}, W, W^{fb}), global parameters (e.g. spectral radius), connection density, adapting slopes of the activation function $f(.)$ and even training W^{out} when no target signal is available. Other interesting methods include *EvoLino*, evolving hidden connections to gradient-based long-short term memory (LSTM) RNNs [161], and using *Neuro-Evolution of Augmented Topologies* (NEAT) as a meta-search algorithm for evolving ESNs [31]. All have shown great potential, highlighting the performance increases and optimisations reservoir pre-training can create for specific tasks. However, evolution is not the only suitable heuristic, others are still (as of 2018) under investigation for pre-training.

2.3.8 Reservoir Computing with Unconventional Substrates

Any high-dimensional dynamical system with an observable state $x(n)$ that is a result of input $u(n)$ could form the basis of a reservoir according to [119]. This implies that any material that can exhibit sufficient dynamics and a fading memory could, at least theoretically, be adopted as a reservoir.

In recent years, this redefinition has marked a mass shift in the reservoir computing community, leading to a “zoo” of possible reservoir computing substrates. With each system the structure and methodology of the RC paradigm is exploited, separating the physical system into two constituent parts: the “reservoir” (the dynamical system) and the interface mechanism (the input and readout).

A novel example of this migration, and an excellent analogy of the reservoir computing paradigm in general, was first demonstrated with the LSM model, where the “liquid” (reservoir) was physically instantiated by a bucket of water [55]. In that experiment, by applying an external stimulus one could identify dynamic states encoded in the ripples on the surface of the water. Then, using these states as the basis of a trained system the authors could exploit the resulting wave interactions to solve logic gate and classification problems.

A decade later, a variety of new physical reservoir computers were proposed. For example, based on optoelectronics (using delayed feedback) [10, 104], memristive devices [102, 21, 29], and non-CMOS devices [99].

Two systems that demonstrate this trend towards unconventional substrates are silicon-based photonic chips (based on nanophotonic crystal cavities) [58, 192], and Atomic Switch Networks (ASNs) [171, 183, 184].

The *photonics chip* primarily exploits the propagation of light through silicon. Inside these chips are photonic crystals that remove the propagation of certain frequencies of light, known as the band gap. Adding a line defect to a crystal produces a photonic crystal waveguide, effectively a process by which light is forced between the defect. Cavities are then created along the line defect to create an optical "resonator" which traps light, increasing the power inside the resonator. These resonators then form an optical reservoir which can be trained and manipulated using different types of readouts. For example, [192] creates a linear system, and the non-linearity is added at the readout through the inherent non-linearity of the measuring equipment. Methods such as this propose an interesting optical option for hardware-based reservoir computing.

The *atomic switch network* approach focuses on the electrical and chemical properties of a material. These networks attempt to mimic the vast complexity, emergent dynamics, and connectivity of the brain. Highly-interconnected networks are constructed by bottom-up self-assembly of silver nanowires. Through a triggered electro-chemical reaction, coated copper seed nucleation sites spawn large quantities of silver nanowires of various lengths, from nano- to millimetre scale. Large random networks are formed, creating crossbar-like junctions between nanowires, and when exposed to gaseous sulphur create $Ag|Ag_2S|Ag$ nanoscale metal-insulator-metal (MIM) junctions. Applying external activation (a bias voltage) to these junctions creates a temporary resistance drop, leading to functional memristive junctions called Atomic Switches. Applying this construction and functionalisation process the ASN method offers some unique properties, such as scalability and practicality in creating highly-complex nanoscale substrates.

The emergent behaviour and dynamics of ASNs are observed through fluctuations in network conductivity, a result of spontaneous switching between discrete metastable resistance states, where locally excited regions cause cascading changes in resistance throughout the system. As such, the non-linear responses to resistive switching are reported to result in higher harmonic generation (HHG), also suggested as a useful measure for quantitatively evaluating reservoir dynamics [171].

Many more physical substrates and reservoir computers exploiting physical interactions exist and have emerged since. These include light-stimulated nanoparticle networks [197]; memcapacitive [190]; quantum [151, 59]; spintronic [189]; carbon nanotube-based [40, 186]; soft robots [144]; tensegrity structures [28]; magnetic [155]; and other memristor-based reservoirs [51]. A recent detailed review of physical reservoir computers, categorised into groups, is given in [185].

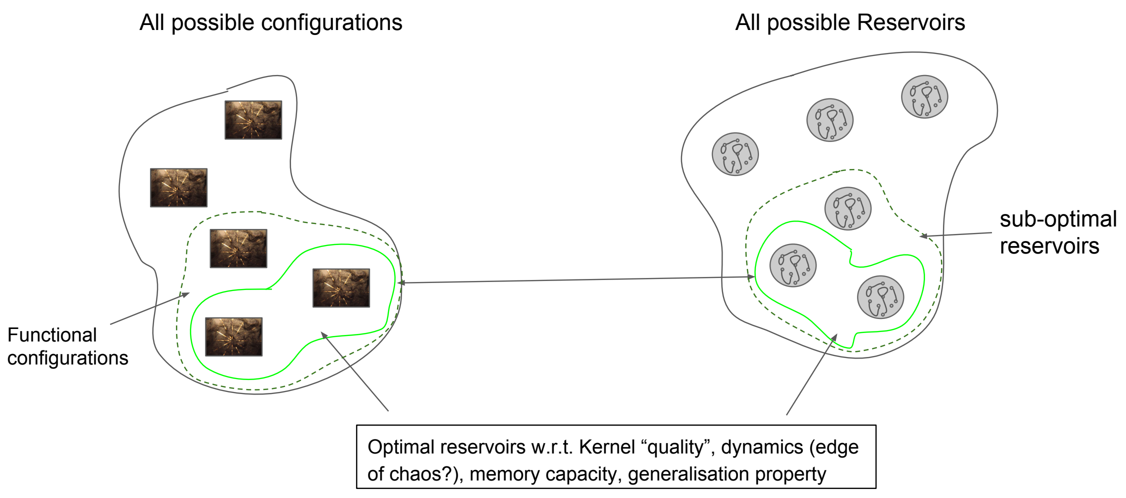

All possible configurations
All possible Reservoirs
sub-optimal reservoirs
Functional configurations
Optimal reservoirs w.r.t. Kernel "quality", dynamics (edge of chaos?), memory capacity, generalisation property

Figure 2.5: Applying the reservoir model we argue that any material, and subsequent configuration, can represent reservoirs of varying quality. Pre-training through configuration is therefore used to find and discover functional/useful reservoirs within all possible material states. We also argue that there is a region that optimal reservoirs exist where convenient informational measures ("metrics") perform best.

2.3.9 Summary

A key advantage to using the reservoir computing paradigm is that there is no requirement to control individual elements within the system, instead, the training process focusses on extracting observable macroscopic properties. This makes it applicable to many complex structures where the exact arrangement or manipulation of internal elements is either too time-consuming, too delicate/complex to implement, or impossible to achieve. However, to determine whether a substrate is computationally exploitable depends upon both underlying physical characteristics and the observable phase space.

In some cases, a physical substrate is computationally useful only when configured or perturbed, as hypothesised and shown in evolution *in materio* (section 2.2). Therefore, to form a useful reservoir that provides a high-dimensional expansion of the input(s) may require the tuning of physical parameters. This itself implies that a single substrate typically realises a variety of reservoir computers through different physical configurations.

In Fig. 2.5, a conceptualisation of all possible configurations and their reservoir equivalence (in terms of reservoir "quality") is given. Using this view, we argue that through physical perturbation and computer controlled evolution some physical substrates can produce reservoir configurations that ideally fit specific tasks. And perhaps, some may be able to realise a wide selection of reservoirs across the quality spectrum, leading to more generic/universal physical computing substrates.

To interpret a physical substrate as a reservoir, we define that the observed reservoir states $x(n)$ form a combination of the substrate's implicit function and its *discrete* observation:

$$x(n) = \Omega(\mathscr{E}(W_{in}u(t), u_{config}(t))) \tag{2.16}$$

where $\Omega(n)$ is the observation of the substrate's macroscopic behaviour and $\mathscr{E}(t)$ the continuous microscopic substrate function when driven by the input $u(t)$ and the input weight matrix W_{in}. The variable $u_{config}(t)$ represents the substrate's configuration, whether that be through external control signals, an input-output mapping, or other method of configuration.

This formalisation of the reservoir states characterises each contributing part of the entire system, including the observation and configuration method, that as a whole represents a physical reservoir computer.

In the next chapter, this formalisation is applied to a uniquely interesting and hard to model physical substrate consisting of carbon nanotubes – similar to those used in the NASCENCE project. The objective is to explore the configuration/reservoir space utilising computer-controlled evolution to solve benchmark tasks.

Chapter 3

Evolving Reservoir Computers

3.1 Carbon Nanotube Reservoir Computing

Reservoir Computing is presented as a useful general theoretical model for many dynamical systems. Here, the first steps to applying the reservoir model as a simple computational layer to evolvable hardware is shown. It is argued that many physical substrates can be represented and configured into working reservoirs given some *pre-training* through evolutionary selected input-output mappings and targeted input stimuli.

This chapter investigates the use of computer controlled evolution (CCE) to configure single-walled carbon nanotube and polymer composites for reservoir computing. Through evolution, a form of pre-training is performed on the dynamical system – which might not necessarily be a natural reservoir candidate – into a functional and optimisable reservoir computing system. This is demonstrated on two temporal reservoir computing tasks: the Nonlinear Auto Regressive Moving Average task (NARMA) and the wave generator task, each requiring different internal characteristics.

3.1.1 Carbon nanotube-polymer composite

The materials under investigation were fabricated within the NASCENCE consortium [18]. The earlier NASCENCE project aim was to investigate candidate materials and techniques for configuring materials for computation. It was concluded that carbon nanotube and polymer composites exhibited interesting information processing capabilities tunable

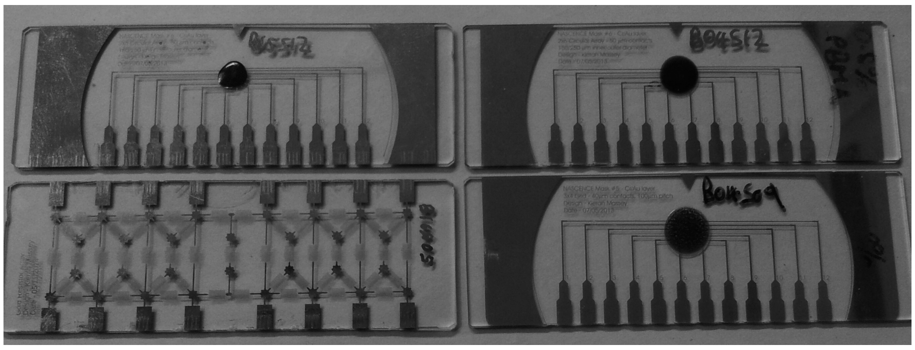


Figure 3.1: Substrates under test. Top left, SWCNT/PBMA mixture with a concentration of 1% SWCNT by weight. Top right, SWCNT/PBMA 0.53%. Bottom left, gold resistor array. Bottom right, SWCNT/PMMA 0.1%

through evolution.

In this experiment three carbon nanotube-based test subjects with varying compositions are evaluated (see Fig. 3.1). In addition, two system "settings" (*short-circuit* and *open-circuit*) and a resistive network are evaluated. The intention is to determine how much the system as a whole is being evolved and to demonstrate the computational capability using only resistivity.

Test subjects one and two are single-wall Carbon Nanotube (SWCNT)/ polymer mixtures with SWCNT concentrations of 0.53% and 1% (by weight) mixed with poly-butyl-methacrylate (PBMA) dissolved in Anisole. Test subject three is a 0.1% SWCNT mixture with poly-methyl methacrylate (PMMA).

The application process of each entails approximately 20ml of the mixtures being deposited onto glass slides with etch-patterned electrodes. Heat is then applied ($\sim 100\,^{\circ}$C for 1hr) to evaporate the anisole and dry the solution in place. To maintain structural integrity, the heating equipment is switched off and the material is left to cool slowly. In the final result, a thick film is left covering the electrodes.

The electrodes are patterned onto glass through photo-lithography. Each glass slide has 12 chromium/gold-contact (40 to 50 µm contact diameter and 100 to 150 µm contact spacing) micro-electrodes arranged in either a circle or square array.

The random formations and settling of SWCNTs within the samples fluctuate considerably. Conductivity of each material is determined by SWCNT density and electrode contact. The heterogeneous behaviour of the material results from the dielectric properties of the polymer and the shifting electronic properties of networks formed from both semi-conducting and metallic SWCNTs.

The electronic properties of the dispersed nanotubes are approximately one-third metallic nanotubes and two-thirds semiconducting nanotubes. The relative size of the nanotubes (100nm to 1000nm length and diameter between 0.8nm and 1.2nm) is significantly less than the gap between the electrodes (between 100 to 150 µm), suggesting that the nanotubes form conductive pathways between the electrodes. The polymer mixed with the nanotubes acts both as a dielectric and as a suspension for the nanotube network. More information on these disordered materials can be found in [132].

Test subject four is a reference material: a gold resistor array patterned onto a glass slide with multiple connection points using etch-back photo-lithography. The resistor array is arguably simpler and more stable with known internal resistance values. This test subject is used to determine whether the technique can be applied to linear mediums and what, if any, are the advantages of SWCNT-based materials over simple resistive networks.

The *open* and *short* circuit settings are applied to verify the significance the material has on the evolvability of the system, that is, to pinpoint what is doing the computation. In the open-circuit no material is connected; the system is simply left to find a solution through system noise, or from unknown characteristics within the system. The conductive sheet (copper tape) is used as a short-circuit connection to assess if the material has any advantageous properties beyond conductivity.

3.2 Hardware Setup

The interface, recording and stimulation equipment used in this experiment forms a hybrid digital/analogue hardware loop using off-the-shelf Data Acquisition (DAQ) Cards.

Computer controlled evolution (CCE) is performed in the digital space on a connected desktop PC using a MATLAB interface. In the analogue/physical space, the material is stimulated using a National Instruments Data Acquisition Card (NI PCI-6723) supplying analogue output signals, which can be routed to any of the electrodes interfacing the material via an Analog Devices (AD75019) 16×16 analogue cross-point switch. An NI

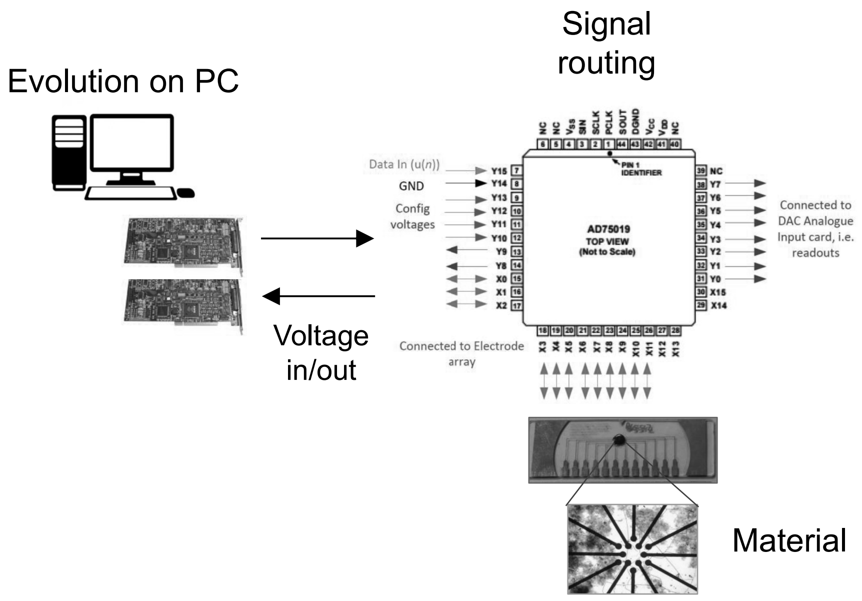


Figure 3.2: Hardware Interface. PC stimulates material via routing matrix.

DAQ card (NI PCI-6225) is used to record analogue inputs from the electrode array via the cross-point switch in the same manner.

The cross-point switch designates which electrodes and DAQ card channels are currently in use and what role each electrode performs. When the evolved configuration is registered on the cross-point switch, bidirectional communication is established between both DAQ cards and the electrodes. An overview of the system is presented in Fig. 3.2.

3.3 Training and Evolution

Within the earlier NASCENCE project multiple control signals were investigated, such as complex signals like evolved square waves [123, 147]. However for this experiment, control signals are restricted to static voltages to avoid any possible interference, or artefacts, that evolution may create in respect to temporal tasks.

The electrical configuration of the substrate is exclusively carried out through the placement and adjustment of static control voltages. The aim is thus to configure the internal characteristics of the substrate by perturbing the natural dynamics, conductivity and signal processing abilities, for example, by creating local or global biases through targeted control voltages.

3.3.1 Evolutionary Algorithm and Representation

The encoding of the configuration is represented by a 21-gene genotype. Every gene is open to mutation and subdivided into: electrode assignment (genes 1–12), redundant genes (genes 13–16), values of static input voltages (genes 17–20) and input scaling on $u(n)$ (gene 21). Genes 1–16 are integer values, all other genes are floating point numbers with a precision of 4 decimal places. Genes 17–20 range between [-5V,5V] and gene 21's range is either [0V,2V] for the NARMA task (already pre-scaled by factor of 10) or [0V,5V] for the wave generator task.

The genome design allows evolution to decide both the number of readouts and the number of static input voltages the substrate is subjected to. At genotype creation, and under the mutation operator, a maximum of 10 possible readouts (referred to as measurable *reservoir states*, RO_n in Fig. 3.3) are possible. This is due to the input signal $u(n)$ and ground (GND) always being required. A maximum of 4 static control voltages can also be applied simultaneously. This feature (implemented by redundant genes) allows evolution to converge towards any assignment, such as 6 readouts and 4 configuration voltages, or, 8 readouts and 2 configuration voltages (Fig. 3.3), as long as the required phenotype size of 12 is always adhered to.

The input scaling gene scaling $u(n)$, effectively an input gain parameter, is added as the material may require varying input-data intensities under different electrical configurations. The gene is initialised at the maximum value for the given task, then left to evolve.

The physical instantiation of the genotype – the phenotype – is implemented via the cross-point switch. The interfacing equipment is set so that all accessible inputs and outputs are connected to the switch. The switch then directs which DAQ channels communicate to the electrode array through the values in genes 1–12, realised in hardware by a 256-bit digital input (SIN) to the switch.

3.3.2 Training cycle

The implemented genetic algorithm applies an elitist 1+λ evolutionary strategy. General parameters of the algorithm are: a population of 5 ($\lambda = 4$), 150 generations, 10 repeat

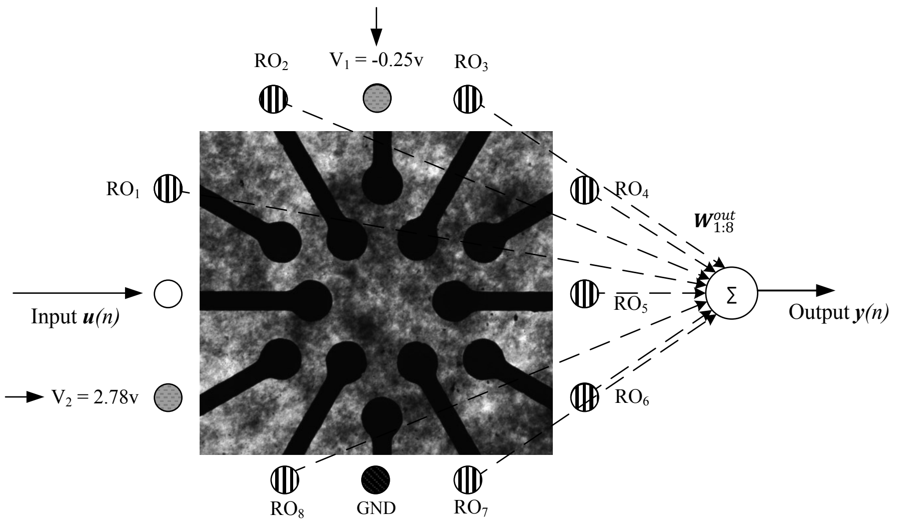


Figure 3.3: Physical reservoir representation using electrodes. Each assigned readout electrode (RO_n) forms the reservoir state. The configuration voltages (V_n) location and value are decided by evolution. The W^{out} matrix is calculated and applied in the digital domain.

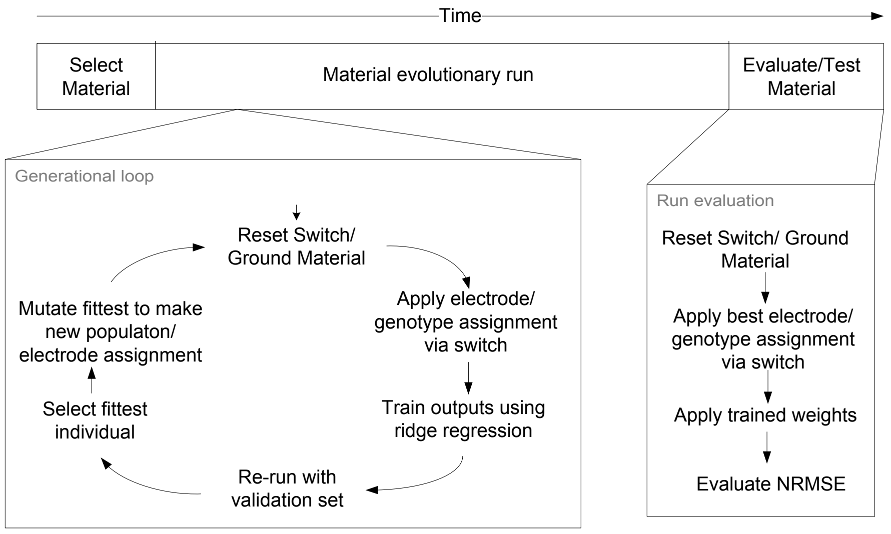

Time
Select Material
Material evolutionary run
Evaluate/Test Material
Generational loop
Reset Switch/ Ground Material
Apply electrode/ genotype assignment via switch
Train outputs using ridge regression
Re-run with validation set
Select fittest individual
Mutate fittest to make new populaton/ electrode assignment
Run evaluation
Reset Switch/ Ground Material
Apply best electrode/ genotype assignment via switch
Apply trained weights
Evaluate NRMSE

Figure 3.4: Reservoir work flow through time: combined evolution and regression training procedure. Generational loop highlights the switch assignment process and training/validation process. The final evaluation process of the found solution is expanded.

Algorithm 1 Material Training algorithm

```
 1: Initialise population
 2: Calculate fitness/cost of population
 3: for each generation gen to max_gen do
 4:     Determine fittest individual from previous generation
 5:     Designate fittest as parent
 6:     Create Offspring through mutation
 7:     for each offspring i to $\lambda$ do
 8:         Reset cross-point switch                ▷ Ground material
 9:         Load genotype into switch
10:         Run on training set
11:         Train using Ridge Regression            ▷ Store weights
12:         Run on validation set
13:         Reapply saved weights
14:         Assign fitness/cost to offspring        ▷ Store fitness/cost
15: Reapply best configuration and saved weights on test set
16: Calculate final NRMSE value
```

runs. In the elitist 1+λ strategy the λ, referred to as the offspring, are mutations-only of the previous generation's fittest individual. In this setting, only one individual's genes persist between generations. In the rare case that a child has the same fitness as the parent, the child is selected to pass on its genes. This allows evolution to neutrally sweep the search space if no immediate fitness change is present.

Within the evolutionary loop, fitness is measured as the performance of the configuration on an unseen *validation* set. In the first stage of the fitness calculation, the material is stimulated with training data and the materials states (electrode outputs) are collected and used to train the readout. Both configuration and trained readout are then tested on the separate validation set and the error is given as the fitness/cost.

The full procedure is shown in Fig. 3.4 and given in pseudo-code in Algorithm 1. First, a random initialisation of the material is performed. Next, the evolutionary run commences, cycling through the generational loop. This loop comprises: a physical resetting (grounding) of the material; the application of a new switch assignment (material "configuration") for every individual; and the readout training process performed on the electrode outputs. As previously stated, the fitness of every new individual is calculated on the validation set using the Normalised Route Mean Squared Error (NRMSE) between trained outputs and a target signal. The final result, calculated on the best individual found in the evolutionary run, is the NRMSE calculated on another unseen set the test set.

3.4 Benchmark Tasks

3.4.1 NARMA task

The NARMA task originates from work on training recurrent networks [11]. It evaluates a reservoir's ability to model an n-th order highly non-linear dynamical system where the system state depends on the driving input as well as its own history. The challenging aspect of the NARMA task is that it contains both non-linearity and long-term dependencies created by the n-th order time-lag.

An n-th ordered NARMA experiment is carried out by constructing the output $y(n)$ given by eq.(3.1) when supplied with $u(n)$ from a uniform distribution of interval [0, 0.5]. For 5-th and 10-th order systems the following parameters are set: $\alpha = 0.3$, $\beta = 0.05$, $\delta = 10$ and $\gamma = 0.1$.

$$y(n+1) = \alpha y(n) + \beta y(n) \left(\sum_{i=0}^{\delta} y(n-i) \right) + 1.5u(n-\delta)u(n) + \gamma \tag{3.1}$$

The NARMA-10 task is a widely accepted benchmark applied to many different reservoir architectures. This makes it an ideal comparison task; however, the task is rarely attempted with reservoir sizes as small as 12 nodes, which is the maximum number of electrodes available here. Therefore, an exact comparison is challenging when comparing to the literature, particularly on reservoirs quoting upwards of a few hundred reservoir nodes.

The NARMA-5 task is presented as its considered a simpler task to the 10th-ordered, because it requires less memory. This was chosen to determine what level of task complexity the material (under its current limitations) can comfortably operate within.

3.4.2 Wave generator task

The wave generator task requires the transformation of an input waveform (a periodic signal) to create new waveforms using temporal features such as phase shifts, delays and harmonic generation. The task is based on Fourier analysis, where different complex waveforms can be expressed as a series of sinusoidal waveforms. The task was first presented as a benchmark for neuromoprhic Atomic Switch Networks using reservoir computing [171]. The task requires the reservoir to output four separate waveforms simultaneously, given a single input sine wave. Each trained output waveform, a sawtooth, square, cosine and sine wave double the original frequency requires different harmonic and phase properties. For example, the square wave requires only odd harmonics, the sawtooth requires both even and odd harmonics, the cosine requires a 90 degrees phase shift, and the double frequency sine wave requires only the first harmonic.

The general performance of Atomic Switch Networks on this task are reported in [171, 183] with input frequencies around 10Hz. The most recent results show task performances using a 64 electrode set-up in [47].

In the following experiments an input frequency of 1 kHz is selected, rather than 10 Hz, as evidence suggests SWCNT/polymer materials tend to produce more interesting dynamics at higher frequencies [146].

3.5 Results

3.5.1 Control Substrates

The first notable result shows that the materials-under-test, when evolved and configured, outperform both the conductive sheet and the open system. This implies the material has a significant impact on the overall computing system.

The performances of each test material are shown in Fig. 3.5 for the NARMA-5 task and Fig. 3.6 for the NARMA-10 task. For the NARMA-5 task the resistor network produces very similar results but with smaller variance. This variance however increases when moving to the harder NARMA-10 task. In some cases, and for NARMA-5 in particular, the SWCNT materials outperform the resistor array but overall appear to have no

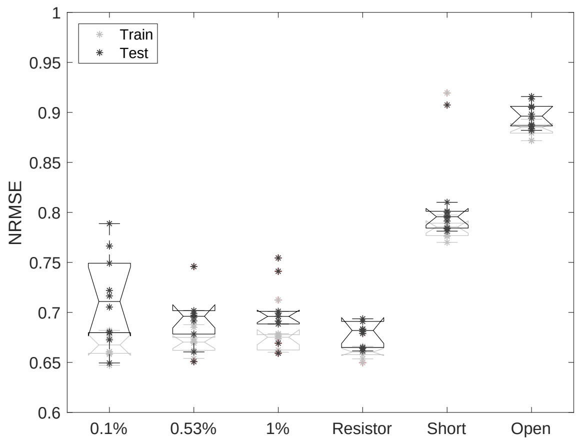


Figure 3.5: NARMA-5 plot of train and test error of 10 runs across all materials (lower error is better fitness).

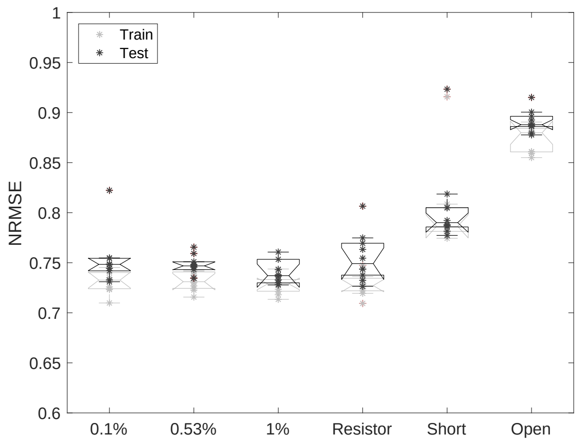


Figure 3.6: NARMA-10 plot of train and test error of 10 runs across each material.

significant difference between them. This suggests at this stage, and on these tasks, the materials possess little more than resistivity, with little or no memory present. And as a result, they perform equally poorly on the two tasks.

3.5.2 Nanotube Density and Stability

Three nanotube concentrations are examined in this experiment. According to the results in Fig. 3.5 and Fig. 3.6 nanotube concentration appears to have a minimal effect on performance for the NARMA tasks.

As seen on the NARMA-5 task, the 0.1% has more variance in its test error, suggesting configurations are potentially less stable than the other two. For example, output weights appear to be severely effected by noisy output states, causing over-fitting in the training error and poor performance on unseen data.

For the NARMA-10 task, the variance in general almost halves. It is hypothesised here that the general poor performance of all configurations explain this result. If training struggles to isolate enough distinct and interesting output signals little improvement or loss in performance will be present.

Stability of configurations is a common problem with this material and setup. In many cases, reapplying the same configuration can result in different output errors. This is referred to here as *solution drift*. To try minimise drift, steps such as the grounding/resetting of the material occur frequently. However, aspects of the material's physics, such as random charge pathways, variable *hopping* and temperature-effected potential barriers [132] are problematic features that are always present. A simple solution to improve accuracy and remove drift is to collect multiple readings and average them, or disregard configurations with large drift enforcing a selection pressure towards more stable configurations.

3.5.3 Memory Capacity

The required memory capacity (MC) of each task correlates to an input lag on the two NARMA tasks. After every evolutionary run, each material is reconfigured with the fittest individual and memory capacity measured (using the measure given in section 8.3.3). The memory capacity of each configuration, for each task, is shown in Fig. 3.7. The

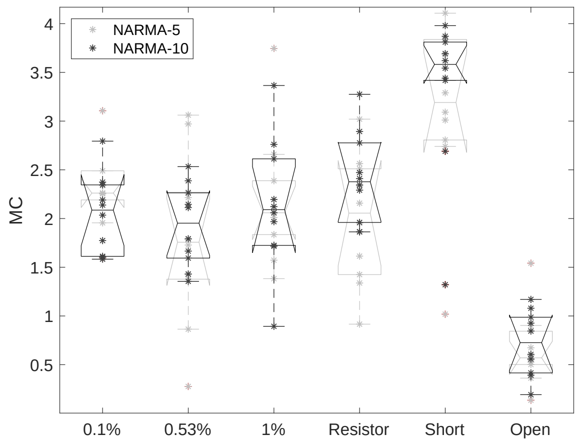


Figure 3.7: Memory Capacity of all test subjects post-evolutionary configuration, across 10 runs.

measured MC changes only minimally and not significantly, despite the difficulty of the tasks. This could be attributed to a number of possible factors. For example, the material cannot increase its MC given the small number of readouts available. However, MC should not be limited by the number of readouts as the internal structure and dynamics do not possess the same limitations. An alternative conjecture is that the method that evaluates the material's memory capacity is too susceptible to noise and is therefore why very low values are produced. Ultimately, the simplest explanation is that the materials simply do not possess the right characteristics to retain a significant memory at the current input-output time-scales, e.g., the material's time-scale is much faster than the recording equipment.

The open system has a very small MC but the conductive sheet appears on average to possess a consistently larger MC. This puzzling result (later explained in Chapter 6) makes it somewhat challenging to determine what significant behavioural differences are at play between the test subjects and the conductive sheet when observing memory capacity alone.

3.5.4 Harmonic Generation

Results of the wave generator task are shown in Table. 3.1. As with the NARMA tasks, similar trends are present; the conductive sheet and open system perform poorly but the resistor array produces similar performance ranges. The PBMA (0.53%) material

Material	**Sawtooth**	**Cosine**	**Square**	**Sine (2t)**
	Avg(Std)/Best	**Avg(Std)/Best**	**Avg(Std)/Best**	**Avg(Std)/Best**
0.1%	0.487(0.11)/0.347	0.079(0.06)/0.019	0.293(0.04)/0.245	0.767(0.36)/0.242
0.53%	4.358(6.85)/**0.325**	2.915(5.60)/**0.004**	2.074(2.89)/0.289	8.986(20.1)/0.255
1%	0.569(0.24)/0.373	0.069(0.08)/0.022	0.308(0.10)/**0.213**	0.881(0.69)/0.326
Resistor	0.499(0.11)/0.365	0.038(0.03)/0.005	0.382(0.06)/0.261	0.705(0.36)/**0.195**
Cond. sheet	3.262(8.65)/0.460	1.025(1.98)/0.20	3.619(10.1)/0.353	0.895(0.47)/0.609
Open sys.	0.750(0.05)/0.697	0.121(0.02)/0.102	0.669(0.08)/0.579	1.000(0.00)/0.999

Table 3.1: Wave Generator results for an input 1 kHz sine wave. Test Error is given for 10 runs.

produces the best task performances averaged across all waveforms. However, across the 10 runs more poor solutions are found compared to the other materials – again suggesting a possible configuration stability issue.

To visualise how well the substrates perform the task, trained outputs of the configured PBMA (0.53%) material are displayed in Fig. 3.8 relative to target outputs. Visually, we can see the performance difference across the waveforms, and in particular, the increased difficulty experienced on the sawtooth task.

3.5.5 Comparison to other Substrates

An exact comparison cannot be made, however an estimation of how well the system performs on the NARMA tasks, despite restrictions, can be made when compared to other reservoir systems. For an optoelectronic reservoir [153] consisting of a 50-node psuedo-network and trained on the NARMA-10 task an NRMSE $\approx$ 0.41 can be reached, and various sized simulated-reservoirs fluctuate between an NRMSE of 0.4 and 0.9 in [196]. The best performances found here are NRMSE's $\approx$ 0.72. This is not considered competitive, however the physical restrictions and how many trainable states are available suggest the materials perform modestly in comparison.

On the wave generator task, the materials appear to be very competitive. In Table. 3.2, the best configuration found with the PBMA (0.53%) material shows lower task errors

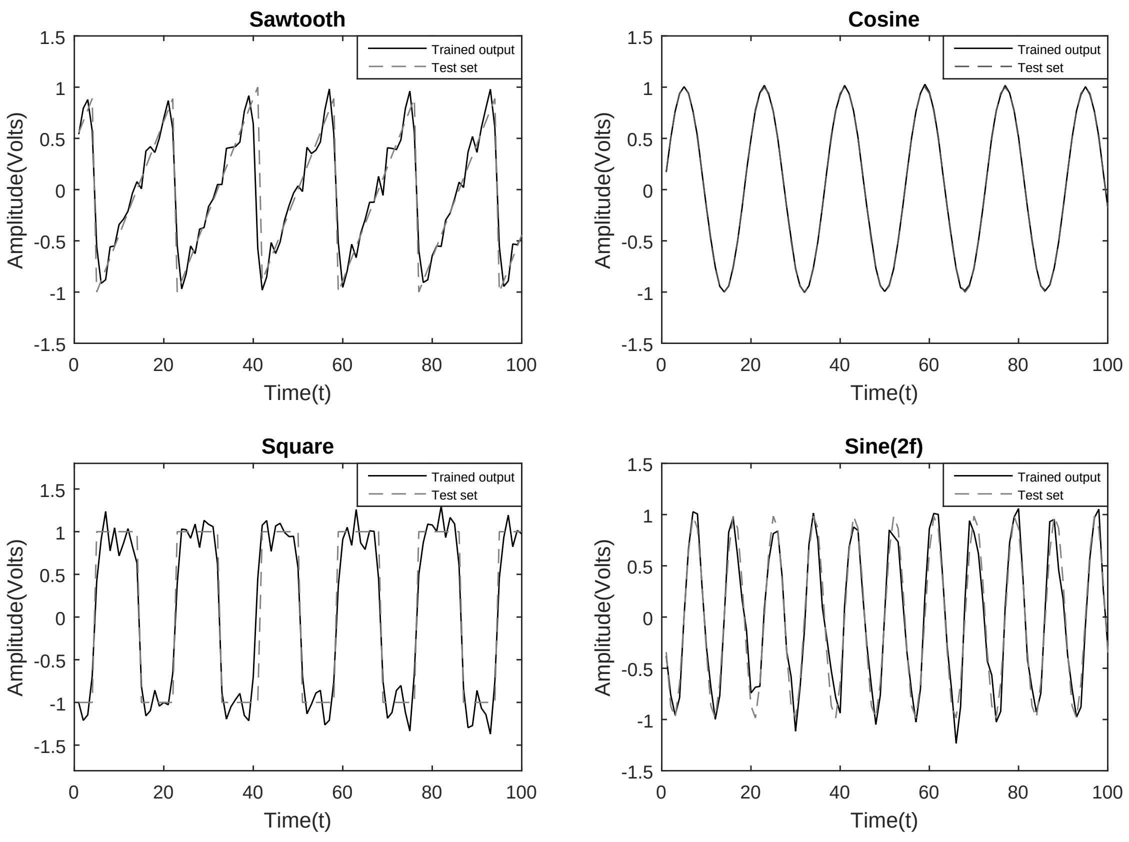


Figure 3.8: Trained reservoir output (PBMA 0.53%) plotted against the desired output for each waveform. All waveforms are trained and outputted simultaneously.

Material	**Sawtooth**	**Cosine**	**Square**	**Sine (2t)**
ASN	0.1071	0.0028	0.0451	0.0910
PBMA (0.53%)	**0.0352**	**0.0001**	0.0830	**0.0325**

Table 3.2: Best wave generator results compared to best Atomic Switch Network [183]. Test Error (Mean Squared Error) given for best configuration on best material (PBMA 0.53%).

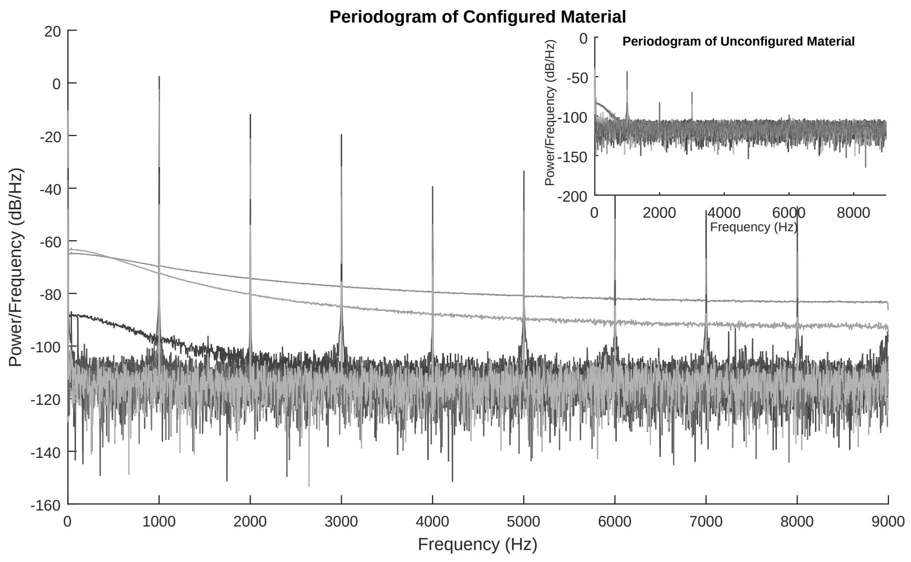


Figure 3.9: Power Spectral Density of readout states on the wave generator task for the PBMA 0.53% material. The main plot shows an increase in harmonic behaviour when the material is “configured”. The subplot shows the unmodified material given an input 1 kHz sine-wave.

than the Atomic Switch Network.

3.6 Summary

This first investigation shows that small configurable (analogue) devices can be trained as reservoirs to tackle difficult system modelling and temporal tasks. However, the results leave room for improvement. The NARMA-10 task results are modest in comparison to optoelectronic reservoirs that use much larger reservoirs (50-nodes and more) and a ingenious reservoir encoding: representation through a pseudo network using pre/post-processing and a long delay line, overcoming the memory problem. For the wave generator task, the configured materials are competitive with or outperform Atomic Switch Networks in [171]. When compared to control subjects (open system and conductive sheet), the carbon nanotube-polymer composites are shown to be computational interesting and central to the computing system.

Despite modest performances, the experiments do show that some materials possess exploitable electrical properties that may not naturally occur without targeted stimulation.

Fig. 3.9 highlights this, showing increased harmonic behaviour that occurs only under configuration: the sub-plot shows only the first three harmonics occur when unconfigured, versus eight or more harmonics when configured.

The biggest limitation in the presented work is the size of the electrode array. The 6 to 10 input electrodes (and hence reservoir nodes in the model) is small in comparison to typical numbers of nodes in simulated and hardware-based reservoirs (often hundreds). With larger electrode arrays there is potential to increase performance, as the training procedure has an increased number of internal states and spatial diversity to exploit.

In the next chapter, further analysis of the framework is conducted focusing on aspects such as how computer-controlled evolution compares to random search and how consistent the solutions are over time. Additional investigations are also conducted to improve performance of the substrate by refining the input encoding.

Chapter 4

Analysis of the Reservoir Computing *in materio*

In this chapter, a more detailed analysis of the reservoir computing *in materio* framework is given. So far, carbon nanotube-polymer composites appear somewhat compatible with the reservoir computing framework when configured through evolution. However, mixed performances have provided little information about whether the current approach is optimal. In the following experiments, four key features are assessed, or implemented:

- The performance of the search algorithm is evaluated compared to random search.
- The input mechanism is adapted to align with the weighted, multiple location input mapping traditionally used in reservoir computing.
- A filter for the observed states is introduced to match the natural time-scales of the material to the time-scales of the task.
- The configuration drift and performance reproducibility is assessed.

At this stage, the first two control subjects (*open* and *short-circuit* system settings) are removed from future experiments. This leaves the three carbon nanotube composites (0.1%, 0.53%, 1% w.r.t SWCNT density) and resistor network.

Results for the above intend to determine whether the observation and configuration methods are suitable, and if they can be tuned further to improve task performance.

4.1 Comparisons to Random Search

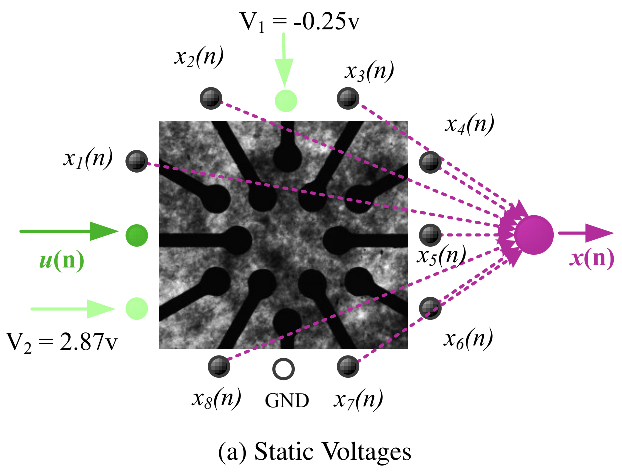
To configure a substrate into a working reservoir a variety of parameters exist, such as the placement of task input-outputs and the placement of control signals and their voltage value. In the previous chapter, the values and choices of these parameters were evolved using an evolutionary algorithm. However, the computational advantage of using evolutionary search over random search has not yet been established. The assumption that computer-controlled evolution is more efficient than random search at creating reservoirs *in materio* is at this stage unproven.

Configuration through random search is documented only once in the evolution *in materio* literature, with Harding's evolvable Liquid Crystal Display [72]. In that work, Harding concluded that using random search alone was not sufficient to create the desired non-linear functions for that particular hardware platform. This assumption carried forward to other platforms within the NASCENCE project with few, if any, comparisons made to random search again. However, conducting a separate investigation into random search for any new computational machine is essential as fundamental differences will be present in hardware, representation, encoding and training methods.

For the first experiment, random configurations are simply compared to evolved configurations. To make a fair comparison, the number of random configurations are equal to the number of evaluations in the genetic algorithm, thus the same number of reservoirs are created and assessed.

4.2 Input Mechanisms

The standard input mechanism applied in the evolution *in materio* technique is to assign each input to a single electrode. In the second experiment, the *one-for-one* input mechanism (Fig. 4.1a) is compared to a *one-to-many* input mechanism (Fig. 4.1b) where the task input is supplied to multiple electrodes on the material, each being multiplied by some *weight*. This technique is more typical of the traditional reservoir computing method, where each input source is connected to the network via an input weight matrix W_{in}. No experimental data or intuition on this type of input mechanism is discussed in the



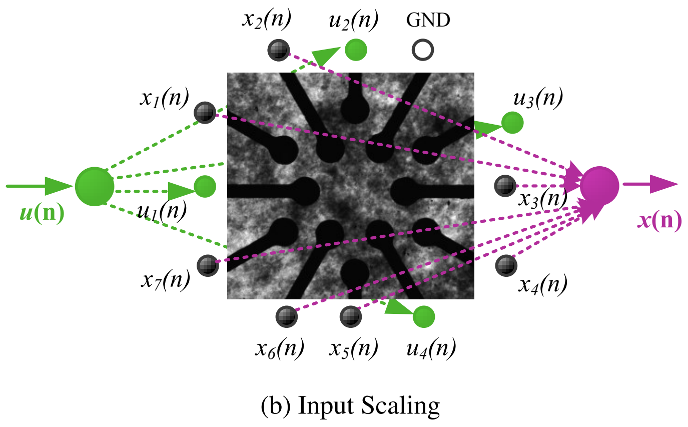


Figure 4.1: Example input mechanisms; a) configuration through static voltages, and b) using multiple weighted inputs to various locations on the array. For the input-weighting scheme, each input $u_i(n)$ is multiplied by a weight stored in the genotype.

evolution *in materio* literature. The hypothesis here is that adding multiple signal sources could promote more complex interactions, activating regions where the material may be electrically weakened, or isolated from the input.

The input weights, i.e. signal gains, are chosen through evolution and are bounded between $[-5\text{V}, 5\text{V}]$. If input-weighting is used, the control signals are not in use. This is due to current hardware limitations on the size of the electrode array (12 electrodes). In total, evolution is restricted to 5 weighted inputs at any one time on the electrode array. In an ideal scenario, using both a weighted input mechanism and controls signals may be desirable, but not realistic on the current size of array.

4.3 Time-scaling

The ability to adjust the temporal response of the reservoir with respect to both the input and desired output can be advantageous. A simple example is to adjust the internal time-scale of the reservoir to the sampling rate at which the data was collected [91]. Time-warping invariant ESNs (TWIESN) do this when sampling from continuous data to discrete data overcoming common time-warping problems within recognition tasks [121]. Multiple time-scaling methods have been investigated for reservoir computing, including input and output re-sampling and time-scaling at precise points within the state collection, e.g. before and after any non-linearity is introduced [163].

For a physical reservoir system, the reservoir substrate will function on a natural time-scale which may or may not be adaptable. Developing a method to match the substrate and task time-scales could offer additional improvements in performance.

In the third experiment, an external *Leak Rate* parameter derived from *Leaky Integrator* Echo State Networks (LI-ESN) [91, 163] is applied and adjusted to control the time-scale mismatch. To fit the practicalities of the system, leaky integration has to be performed after the observation function. Effectively, this turns the leak rate parameter α into a simple adjustable digital low-pass filter, producing a smoothing effect which controls the speed of the reservoir's dynamics. The *filtered* reservoir state is represented as:

$$\tilde{x}(n) = (1-\alpha)x(n-1) + \alpha\Omega(\mathscr{E}(u(t), u_{config}(t))) \tag{4.1}$$

$$y(n) = W_{out}\tilde{x}(n) \tag{4.2}$$

The parameter α has a range between [0,1]; it neither retains, nor leaks beyond the original boundaries of $x(n)$. Low values of the leak parameter a induce slower dynamics coming from the driving input and when time-scaling is not used, i.e. $\alpha = 1$, Eqn. (4.1) reduces to Eqn. (2.16). The final output of the filtered system is given in Eqn. (4.2).

Gene No.	1	2	3	4	⋯	12	13 ⋯ 16	17 ⋯ 20	21	22
Info.	12	8	13	4	⋯	10	7 ⋯ 14	2.435 ⋯ −2.945	5	1

Figure 4.2: Example Genotype. Any gene within the first 16 genes, ≥13 represents a configuration or weighted input. Genes 17 to 20 represent the static voltage input or input signal gain depending on the input mechanism in use.

4.4 Training Process and Encoding

The genotype representing the electrical *configuration* is extended to 22 genes in the following experiments. The first 12 genes hold the functional role of each electrode on the array, i.e. whether an electrode is an input, output, or an additional control signal/weighted input. An additional 4 *inactive* genes are added to allow evolution to dynamically select the size of the reservoir, i.e. the number of material states in use. The next 4 genes represent the floating-point values for each control signal, or, the *weight value* if the input-weighting mechanism is used. The final two genes in the genotype hold the floating-point value for the time-scaling parameter (α) and the weight value for the one input that is always required. An example genotype is given in Fig. 4.2.

The previous (1 + 4) evolutionary strategy (ES) is applied again. This is compared to random search, where each configuration is a random initiation of the genotype selected from a uniform distribution of the minimum and maximum values possible for each gene.

For both search methods, a maximum of 750 fitness evaluations are conducted per run, for 10 runs. The limited number of generations and runs was decided upon as each set of 10 runs typically takes 5 hours to complete for each material. What was leaned from the previous experiments is that 150 generations is, in general, more than adequate to find good solutions, and 10 runs provide some meaningful statistics.

The overall training process for the three experiments breaks down into four key stages:

1. **Reservoir Creation**: The material configuration is loaded onto the cross-point switch, establishing communication between the DAQ cards and the material. The material is then stimulated and the output response is trained.

2. **Reservoir Selection**: The *validation set* error of each individual is compared. The fittest reservoir is selected and compared to the global best solution. If the error is below the global, the new reservoir becomes the parent and passes its genotype onto the next generation.

3. **Create a New Population**: A new population is created from the parent reservoir using a random mutation. The applied mutation operator depends on a mutation probability assigned to: a one-for-one swap between active genes (20%), replace an active gene with an inactive gene (20%), or, adjust the value of the control signal/input weight/time-scaling parameter α (60%) using Gaussian noise (bounded by the min/max voltage range).

4. **Final Reservoir Assessment**: Once evolution is completed, the global best configuration and trained readout W_{out} is reloaded and reapplied. The material is then evaluated on the *test set*, giving the final reported NRMSE. The last stage tests the reservoirs generalisation to new data and its configuration stability, i.e. its repeatability and consistency of behaviour to the same configuration and stimulus.

4.4.1 Benchmark: Santa Fe Time-Series Prediction

A new time-series prediction benchmark is introduced here that requires different computational features from the previous two benchmarks. The task is to predict the next value of the Santa Fe time-series Competition Data (dataset A)[1]. The dataset holds original source data recorded from a Far-Infrared laser (FIR) in a chaotic state.

In the training process the first 5000 values of the dataset are used. This is sub-divided into three sets: 2500 values for the reservoir weight training process (training set), 1250 for the evaluation of each trained reservoir (validation set), and 1250 values to re-evaluate the final evolved reservoir (test set). The first 50 values of each sub-set are discarded as an initial washout period.

To demonstrate the reservoirs role a simple evaluation of task complexity was conducted. When comparing the original input and output of the test set, i.e. $E(u(n), y_{target})$,

[1]Dataset available at [200] and directly through MATLAB's Neural Network Toolbox Sample Data Sets: http://uk.mathworks.com/help/nnet/gs/neural-network-toolbox-sample-data-sets.html

Material	***Evo* Min.**	***Rnd* Min.**	***Evo* Avg. (Std)**	***Rnd* Avg. (Std)**
0.1%	0.416	0.522	0.443 (.011)	0.651 (.106)
0.53%	0.440	0.519	0.454 (.011)	0.656 (.111)
1%	**0.242**	**0.439**	**0.306 (.056)**	**0.489 (.042)**
Resistor	0.498	0.582	0.536 (.023)	0.756 (.082)

Table 4.1: The minimum, mean and standard deviation of the *test* error (NRMSE) for both search methods across 10 runs.

a *Normalised Root Mean Squared Error* (NRMSE) = 0.9771 was achieved. Using the linear model ($y = W_{out}u(n)$) trained on the target y_{target}, an NRMSE = 0.9241 was achieved. These results imply a significant level of additional processing is required by the reservoir to reduce the NRMSE.

4.5 Results

4.5.1 Random Search

For all materials tested, computer-controlled evolution outperforms random search (see Table 4.1). This confirms that, on average, reservoirs with better performances can be found through evolutionary search. This experiment also shows that the SWCNT/PBMA 1% material tends to outperform the other materials using both random and evolutionary search. On average, even with random search, it still outperforms the best evolved resistor configuration. This result is significant when compared to the results found in the previous chapter, where the performance gap between the configured resistor and the configured materials was smaller than anticipated. The results also suggest improved reservoir performance may coincide with a SWCNT density around the percolation threshold of 1%, as stated in [130].

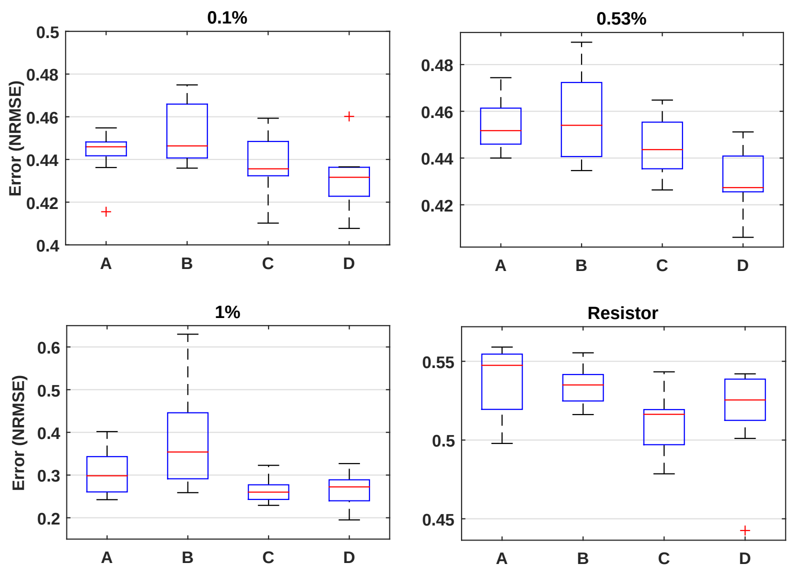


Figure 4.3: Errors using controls, input-weighting and time-scaling: (A) control signals with no time-scaling; (B) control signals and time-scaling; (C) input-weighting with no time-scaling; (D) input-weighting and time-scaling.

4.5.2 Input-Weighting and Time-Scaling

The results given in Fig. 4.3 of the input-weight mechanism and time-scaling feature are mixed. When only applying the input-weighting mechanism, error on average falls compared to control signals and no time-scaling. When combining both input-weighting and time-scaling the *best* runs improve significantly, but the same improvement is not consistent across all runs. Despite only 150 generations being performed, the best case found has an average improvement of 15% and the worse case an improvement of 3%. The most notable performance increase is seen in the SWCNT/PBMA 1% material which already far outperforms the others. This implies both input-weighting and time-scaling can refine and tune both *poor* and *good* performance material reservoirs. However, applying time-scaling by itself does not, on average, always offer an improvement in performance. This suggests some interesting relationship between time-scaling and the input-weighting mechanism exists, that at this point still requires further investigation.

Example evolutionary runs from all materials using both input-weighting and time-scaling are shown in Fig. 4.4. The biggest and most consistent error drops occur with the 1% material. The resistor network on the other hand struggles to reach the same

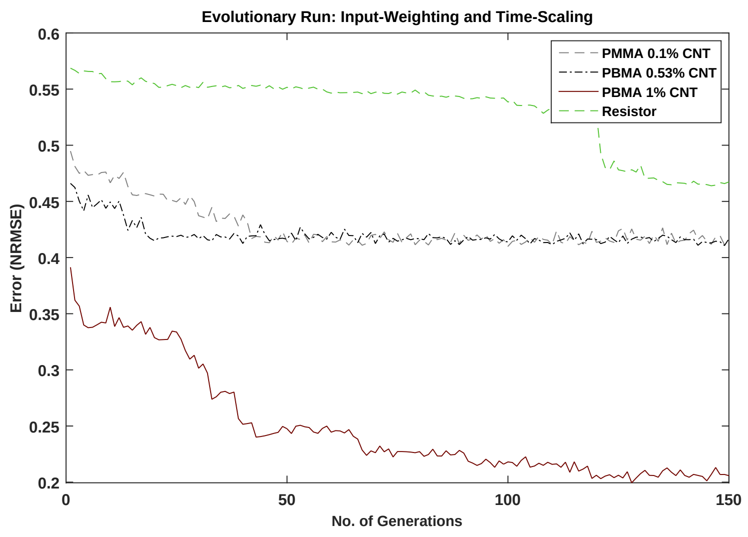


Figure 4.4: Evolutionary runs of each material with both time-scaling and input-scaling applied. Variation in error, both here and in Table 4.1, suggests the search space complex and dependent on nanotube density.

performances of the carbon nanotube substrates, in stark contrast to the previous results on other benchmarks.

4.5.3 Repeatable Performance

A problem with the current system is solution drift and consistent performance with regards to a configuration. In this section, a brief investigation is conducted to measure the materials' change in performance over time.

The material is a passive medium with static structure, therefore reproducing the same behaviour without large fluctuations in performance is plausible. Small fluctuations in performance will however always persist as the carbon nanotube composites respond to voltage stimuli in a non-deterministic manner, for example, just the presence of environmental noise makes the system non-deterministic.

To evaluate the repeatability of performance, three separate sets of re-evaluations are carried out on the best performing reservoir with the best material (SWNCT/PBMA 1%). Each evaluation set consists of 100 reassignments of the configuration, task input and the trained readout after a time period of: i) an hour, ii) one day and iii) one week.

After an hour, the material is assumed to relax back to a natural state. However, the

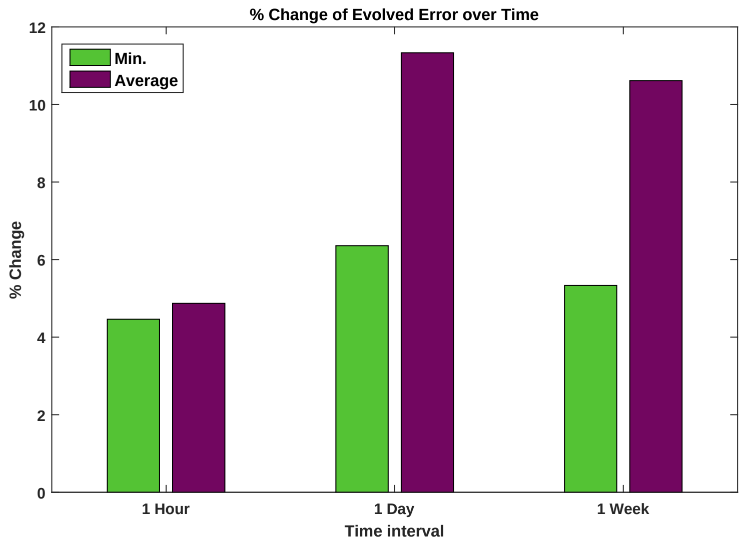


Figure 4.5: A graph of the percentage change in error from the original evolved *test set* NRMSE of 0.195 over different time periods. This is shown for the best reservoir found with the SWCNT/PBMA 1% material using both time-scaling and input-weighting.

environment may have altered, such as room temperature and cross-talk from any electrical equipment close-by. The same is assumed for the one day interval, but in addition, slower structural changes may occur that have longer time-scales. Further "drift" beyond this point is likely to trail-off. In a real-world application, the material may also be reconfigured or evolved to another task, then reassigned to the previous task. To measure these effects, the material was evolved several times over a week to solve a different computational task.

Fig. 4.5 shows the percentage change, defined as

$$\% \text{ change} = \frac{\text{new error} - \text{initial error}}{\text{abs}(\text{initial error})} \times 100 \tag{4.3}$$

between the first evolved error (NRMSE = 0.195) and after each time-period. The results show, on average, an error change of $\sim 4\%$ after an hour, amounting to a difference in NRMSE by ~ 0.0078, which would be considered very small. After a day, the average error increases to $\sim 11\%$, with the smallest change roughly 2% more than after an hour. After a week and several reassignments, average error change settles at 10.6% from the initial NRMSE of 0.195 to 0.2157.

What can be concluded is that solution/configuration drift and performance degradation is an almost certainty, however, it is considered marginal to the overall performance.

Reservoir Type	**NMSE**	**NRMSE**	**Res. Nodes**
Echo State Network (evolved)	0.009	0.098	50
Echo State Network [158]	0.018	0.134	50
Optoelec. (numerical) [145]	0.02	0.141	200
Optoelec. (numerical) [78, 20]	0.022	0.148	200
Mackey-Glass Oscillator [9]	0.023	0.151	50
In *materio* Reservoir	**0.038**	**0.195**	**7**
ESN (evolved and sampled)	0.042	0.205	7 (50)
Echo State Network (evolved)	0.055	0.235	7
Optoelec. (experimental) [78, 20]	0.106	0.326	200
Optoelectronic [104, 9]	0.123	0.35	400

Table 4.2: Comparison table of other reservoir computing systems, with our system highlighted.

Despite being reconfigured and evolved to different tasks, the material still retains similar structural and computational properties to which it was first evolved for.

4.5.4 Reservoir Comparison

The proposed system, in the context of other reservoir systems, shows very competitive results. Table 4.2 shows a comparison between the in *materio* reservoir, simulated/numerical reservoirs and hardware reservoir computers.

Three *evolved* (simulated) echo state networks (ESNs) are also provided. Each evolved network uses the same evolutionary process and number of evaluations as the material. Two variations of these ESNs are also given; two networks where every node is used (i.e. 7 or 50 neurons), and a 50 node ESN with 7 nodes randomly sampled to form the trainable states. This last network provides a brief insight into what effect sub-sampling states from a much larger network has on training and thus performance.

Table 4.2 shows that the SWCNT/PBMA 1% reservoir outperforms experimental optoelectronic reservoirs found in the literature on this task, with a significant reduction in the number of states used. The *in materio* reservoir also outperforms the evolved

7 node ESN and the sampled 50 node ESN.

The relationship in performance between the sampled and non-sampled networks could provide further insight into how the reservoir might scale with more electrodes, e.g. if the same relationship exists, a 50 node/electrode SWCNT/PBMA 1% reservoir could potentially produce an NRMSE < 0.098.

4.5.5 Potential Limitations

In contrast to the output technique used in evolution *in materio*, the reservoir derives its output from the cumulative behaviours of multiple electrodes. The readout layer selectively chooses and separates interesting output signal patterns.

An output mapping that exploits multiple electrodes could have several advantages: (i) form a layer of abstraction that extends the material's programmability; (ii) provide a more robust/fault tolerant output; (iii) represent an output mechanism that can scale with hardware and task complexity; (iv) offer the opportunity to use multiple observation methods to define the task output, i.e. an output could combine electrical, thermal, optical and many more types of observation.

A possible disadvantage to the reservoir implementation however is a desire for more observable states, and therefore a more fine-grained observation mechanism to fully extract the materials computational complexity. With this in mind, the limitations of the 12 electrode system become a great hindrance, not only with regards to performance but also by restricting how the system can be configured.

Another growing problem is the conventional reservoir model, despite its advantages and suitability, possesses limitations: reservoirs deal poorly with *simultaneous* multiple time-scales [119]. Adding a time-scaling parameter cannot overcome this fundamental problem. A number of suggestions and demonstrations as to how this can be solved are discussed in [89, 203], such as creating hierarchical and modular reservoir systems (see Chapter 7).

Implementing an extendible structure in hardware is an intriguing concept for several reasons: not only can it solve issues with time-scaling but it could result in a larger reservoir system with vast reservoir/material *diversity*. This diversity could come from multiple materials exhibiting different reservoir properties. In some sense, a system like this could complete the vision of Miller's high-performance analogue computer made up of evolved materials that form functional primitives [135].

4.6 Summary

The experimental results in this chapter show evolved configurations across all four test substrates consistently produce reservoirs with greater performance than randomly configured reservoirs. Configuring a material without evolution can be inefficient with this system, much like Harding's liquid crystal system.

The results also show that applying both input-weighting and time-scaling simultaneously can provide additional *tuning* to the task, improving performance further. For one material (SWCNT/PBMA 1%), the evolved reservoir outperforms – for this task – all other hardware-based reservoir computers found in the literature. The same material also outperforms a simple evolved simulated echo state network of the same size.

The performance of the material is demonstrated to be consistent both after long time-periods and after reconfiguration to other tasks. Little is reported about performance degradation with time in the evolution *in materio* literature. This brief analysis therefore gives some insight into what to expect when implemented in a real-world application.

In the next chapter, we directly compare the evolution *in materio* method – with its direct encoding – against the framework. To compare the two, we focus on a non-temporal benchmark task and apply both methods to the same identical materials.

Chapter 5

Exploiting Low-level Evolution and High-level Abstraction

In this chapter, a detailed comparison is made between the direct evolution *in materio* approach and the reservoir computing *in materio* abstraction approach.

It can be argued that a direct programming approach is more likely to outperform the highly-programmable reservoir abstraction approach. In support of this argument, Conrad [36] states that: "*a computing system cannot at the same time have high programmability, high computational efficiency, and high evolutionary adaptability*". This can be paraphrased as an increase in programmability and evolvability often results in some loss in performance, forming a basic trade-off problem. In the following experiments, this argument is tested.

Results that disagree with the argument would suggest extra programmability given by the reservoir representation and readout function can provide new levels of exploitation, scalability and complexity to the evolution *in materio* concept without a loss in performance.

In order to compare the two techniques another benchmark task is introduced. At this point all tasks have been temporal because the reservoir computing paradigm takes advantage of such properties. However, reservoirs can also be assigned to non-temporal tasks, if adjusted appropriately, i.e. by reducing or removing the impact of previous states. Almost all tasks applied in evolution *in materio* are non-temporal, set by the output encoding; typically using a buffer with digital voltage thresholds [141, 143]. The few tasks that appear temporal have involved the evolution of robot controllers [74, 138]. However, the chosen encodings quantise the problem into a non-temporal classification task – turning a motor on or off when a set frequency is detected – requiring no information from previous time-steps. Therefore, to make a fair comparison, a non-temporal task has been chosen as the benchmark for this chapter.

In addition to the chapter's main investigation, another comparison is made with evolved *in silico* (simulated) reservoirs, providing further detail on the effects of sub-sampling the reservoir state space. So far, only a brief demonstration has been presented on the effect sub-sampling a large state space has on performance, e.g. Table. 4.2. Sub-sampling is potentially a significant problem with the current setup due to the limited number of observable states from the material.

At the end of the chapter, the internal parameters of the evolved simulated reservoirs are analysed. The intention here is to try determine what possible dynamical traits are present in configured materials, based on similar performance simulated networks. Up to this point, little information has leaked through about general characteristics of the material and its configurations. This was due to previous experiments being focussed on evaluation through specific tasks. However, learning any general characteristics without any systematic task-independent methodology is non-trivial, if not impossible.

5.1 Training

In the following experiment the same $(1+4)$ ES is used for 150 generations, with 20 independent runs. Gaussian mutation operators are added, manipulating the search to act similar to a hill-climber algorithm. This is chosen to reduce the retention effect of degenerative fitness jumps previously experienced by rapidly changing the configuration parameters, most notably found when flipping inputs to outputs and vice versa. However, applying Gaussian adaptation to the $(1+\lambda)$ ES could push training towards local optima. Therefore, to train these materials requires some level of comprise, as the material may not be truly "reset", i.e. the material may retain charge from previous inputs and evaluations, or, the material may have permanently changed its phase space – a significant problem if the underlying material structure is non-stationary.

The task for this chapter is to classify binary classes rather than a time-series output. A threshold mechanism is introduced to translate the trained outputs into binary classes. To evaluate the accuracy of the binary classifier, and to conduct a fair comparison with the previous EiM results, the fitness calculation in [141] was implemented:

$$\text{fitness} = \frac{TP \times TN}{(TP + FP)(TN + FN)} \tag{5.1}$$

where *TP* is the number of true positives, *TN* is true negatives, *FP* is false positives, and *FN* is false negatives.

To select a threshold, a simple optimisation loop is applied (post-state collection). The best threshold found is attached and stored with the configuration and reimplemented at the testing phase.

5.1.1 Benchmark: Fishers' Iris Classification

The Iris dataset[1] (also known as *Fisher's* Iris dataset) is a well-known multivariate classification problem, and has been used to benchmark the classical evolution *in materio* technique numerous times [35, 141, 142]. The objective is to classify three species of the Iris plant given four attributes of petal and sepal length and width.

The task difficulty arises from two classes being linearly inseparable from each other. Although a popular benchmark, it tends to reflect little about how much computation the system can perform as the task is relatively simple, e.g. with a linear system a rough accuracy of 76% is possible.

The Iris dataset contains 150 instances, with 50 instances of each class/species. The only pre-processing of the dataset was to evenly divide it into training and testing sets of 75 randomised instances, containing 25 instances of each class.

Each predicted class of the reservoir is represented as a separate reservoir output with a binary value, and each attribute is represented by a floating-point input voltage.

The input-output mapping of the task inputs $u(n)$, outputs/classes $y(n)$ and observed reservoir states $x(n)$ are given in Fig. 5.1. Four inputs ($u_{1:4}(n)$) are required for the Iris task, with each input multiplied by the input weight matrix W_{in} and applied to a evolved

[1] Dataset provided on the UCI Machine learning repository at: https://archive.ics.uci.edu/ml/

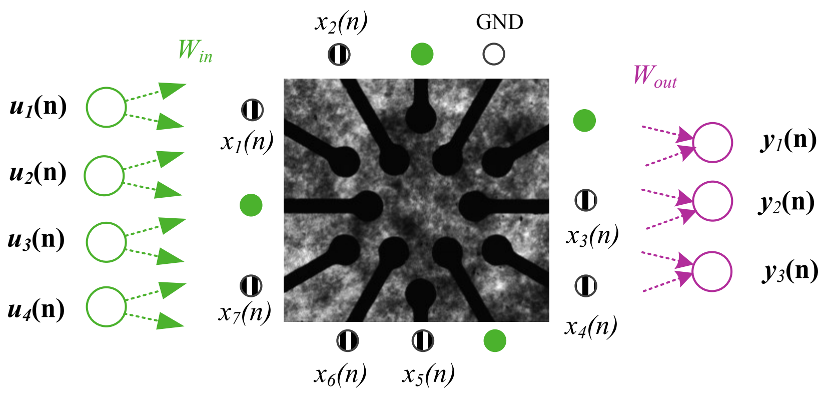


Figure 5.1: Example input-output mapping for Iris task.

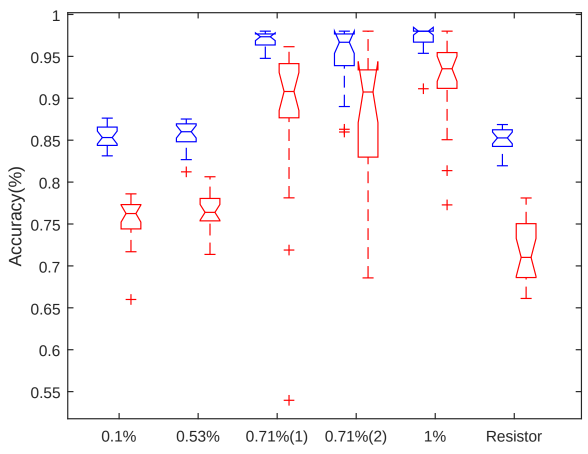


Figure 5.2: Evolved accuracies of each material for the *Iris* task; training accuracy (blue, left) and test accuracy (red, right).

location (green). To form the 3 output classes $y_{1:3}(n)$, the reservoir states (e.g. $x_{1:7}(n)$) are multiplied by the output matrix W_{out}.

5.2 Results

5.2.1 Direct vs. Abstraction

Fig. 5.2 shows the evolved accuracies of each material using the reservoir computing approach. Each material shows training accuracy (blue, left) and test accuracy (red, right) across 20 evolutionary runs. The 0.71%(1) and 0.71%(2) entries refer to two different samples of the same concentration.

The results show the control material (resistor array) has statistically lower accuracies

System : Material	**Training**	**Std**	**Test**	**Std**
EiM [141] : PMMA 0.71%	84.7	—	77.1	—
RCiM : PMMA 0.71% (1)	96.96	0.92	88.15	10.01
RCiM : PMMA 0.71% (2)	94.9	3.92	87.91	7.85
EiM [142] : PBMA 1%	93.33	—	87.87	—
RCiM : PBMA 1%	97.14	1.61	91.90	5.48
CGP [141]	97.7	—	93.6	—

Table 5.1: RCiM compared to EiM. Average (mean) accuracies (in percentages) are shown for training and test data. For RCiM, standard deviation (std) is provided.

than the lower density materials, suggesting material properties beyond resistivity are being exploited.

As the distributions are non-normal, the non-parametric Wilcoxon rank sum test (equivalent to Mann-Whitney U-test) is used to test the null hypothesis H_0 = the medians of the samples are the same. A value $p < 0.05$ indicates statistically significant rejection of H_0 at the 95% confidence level. The non-parametric Vargha-Delaney effect size A [193] is also given (only for $p < 0.05$ in Table 5.2 and Table 5.3). $A > 0.71$ symbolises a large effect size and therefore the significance is of substantial importance and not simply an effect of the sample size.

The rejection of H_0 with a large effect size in Table 5.2 shows that the materials provide a significant contribution to the solution, and that the solution is not dominated by contributions from rest of the system, or from resistivity alone.

Fig. 5.2 shows performance varies with respect to carbon nanotube density, with densities of 0.71% and above producing the highest accuracies. However, Table 5.2 shows no statistically significant difference between the higher concentration materials (0.71% and 1%), demonstrating similar medians and distributions with p-values > 0.05. This suggests once a network of nanotubes is established – effectively determined by nanotube density – an increase in computational performance is minimal. The same hypothesis is discussed in [130].

Table 5.1 shows the comparison between EiM, RCiM and an *in silico* evolutionary

Test materials	***p* U**	**effect size *A***
0.1% / Resistor	0.002	0.78
0.53% / Resistor	0.0001	0.85
1% / 0.71% (1)	0.086	—
1% / 0.71% (2)	0.051	—

Table 5.2: Wilcoxon/Mann-Whitney (U) and Vargha-Delaney *A* effect sizes, between different density materials and the control.

optimisation technique called Cartesian Genetic Programming (CGP) [141, 142]. The reported EiM accuracies in Table 5.1 come from two different hardware configurations: [141] uses Mecobo 3.0, a digital stimulation and measurement board; [142] uses Mecobo 3.5, an analogue stimulation and measurement board.

The accuracies quoted are assumed to be mean values, across 10 runs for [142] and 20 runs for [141]. Standard deviation was not provided for the EiM method and is therefore omitted from Table 5.1. In the comparison experiments, 20 runs are reported with standard deviations.

As shown in Table 5.1, the RCiM method outperforms the EiM technique across both Mecobo platforms. The mean accuracies of both Mecobo platforms fall outside the 95% confidence levels of the RCiM experiments, suggesting the increase in performance is significant. The performances of physical reservoirs also compare favourably to the *in silico* evolutionary programming technique (CGP). The CGP mean accuracy, although higher than the RCiM accuracy, falls within the 95% confidence level of the PBMA 1% material; that is, it is not statistically significantly higher.

5.3 Reservoirs *in materio* vs. *in silico*

For this experiment, three evolvable *in silico* reservoirs (similar to the ESNs used in Chapter 4) are used. The performance of the best material (1% PBMA) is compared to three evolved simulated reservoirs, shown in Fig. 5.3, with training accuracy (blue, left) and test accuracy (red, right) given across 20 evolutionary runs. In this experiment,

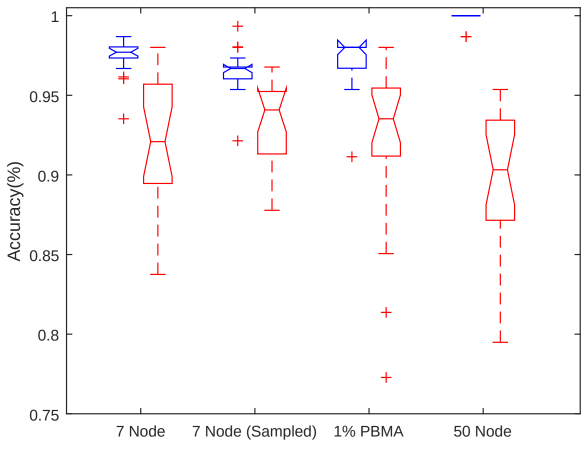


Figure 5.3: Experimental results of three simulated reservoirs and the 1% PBMA material; training accuracy (blue, left) and test accuracy (red, right).

Test materials	***p* U**	**effect size *A***
1% / 7-node ESN	0.51	—
1% / 7-node (sampled) ESN	0.74	—
1% / 50-node ESN	0.0481	0.684

Table 5.3: Wilcoxon/Mann-Whitney (U) and Vargha-Delaney *A* effect sizes, between PBMA 1% material and simulated reservoirs.

two reservoir sizes are compared; a network with 7 internal nodes and two networks containing 50 internal nodes. The 50-node reservoirs are split into different readout types; one where all nodes are accessible to the readout layer and another where the readout can only access 7 randomly chosen nodes. A hypothesis is that when extracting states from the material, only a small subset of the materials state space is sampled – in our case, through roughly 7 electrodes. It is therefore imperative to observe how this affects simulated reservoirs.

The similarity in accuracies found in Fig. 5.3 suggests the material forms a trainable reservoir within a similar performance range of the 7-node and the 50-node sampled reservoirs. As shown in Table 5.3, there is no statistically significant difference in performance when comparing the material to the sampled and 7-node simulated reservoirs. However the 50-node network, with all states in use, is statistically significant for the Mann-Whitney test (i.e. different medians).

Upon further examination of Fig. 5.3, the material appears to be exploiting the benefits of a smaller readout combined with a larger hidden network for this task, much like the sampled network. This is seen with the 50-node network where overfitting is present, but reduced in the 50-node network where only 7 neuron states are sampled to form the output. This *sampling effect*, in the context of the number of electrodes available, is an interesting and possibly advantageous, or limiting factor, that needs to be further understood.

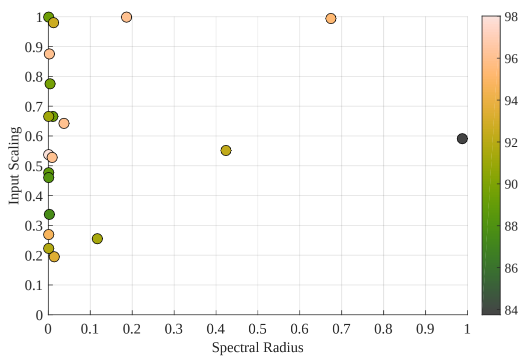
5.3.1 Interpreting Reservoir Characteristics

In the previous section, three evolved simulated reservoirs are compared. The general parameters and structure for these networks are: a sparse reservoir connectivity of 10%; uniformly distributed weights between $[-1, 1]$ for input and reservoir weights; tanh neurons; no feedback weights; and a readout trained using ridge regression.

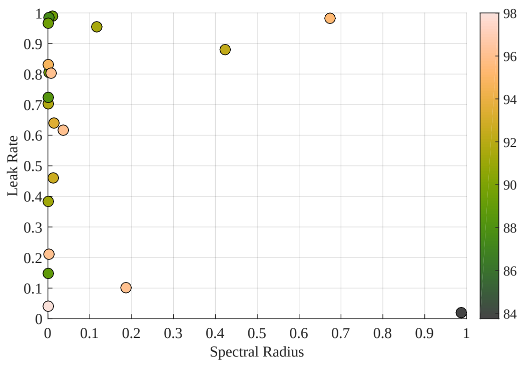
Each reservoir has three key parameters that are selected through evolution; the *spectral radius* scaling, *input scaling* and *leak rate*. Each parameter is adjusted to tune the dynamical behaviour and memory of the reservoir. In echo state networks, the spectral radius determines how fast the influence of the input degrades and the stability of the reservoirs activations. The Input scaling parameter is used to tune the non-linearity of the reservoir and tune the proportional effect the current input has compared to previous activations. The final parameter, leak rate, is used to match the speed of the reservoir's dynamics to the task input and/or output, essentially applying additional filtering to the activation of each node [117].

Fig. 5.4 shows the relationship between parameters and task performance for each evolved network. Each plot displays 20 evolved ESNs with network sizes of; 7 nodes (a & b), 50 nodes (c & d), and 50 nodes with only 7 nodes being used to train/form the output (e & f). The colour map shows the Iris test accuracy of each evolved parameter set.

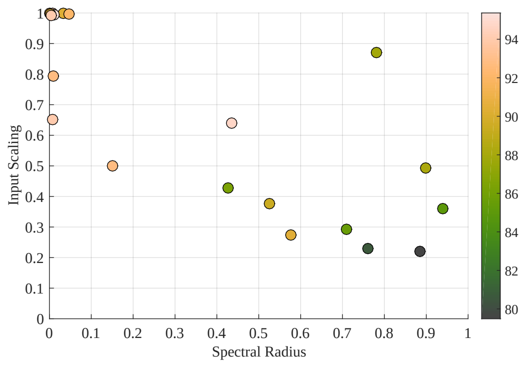
By visualising the evolved parameters and their relative performance, we can determine the desired dynamics for a given task and suggest what dynamics need to be exploited within the material. From the graphs, each network (independent of size) typically possesses the following for the *Iris* task:



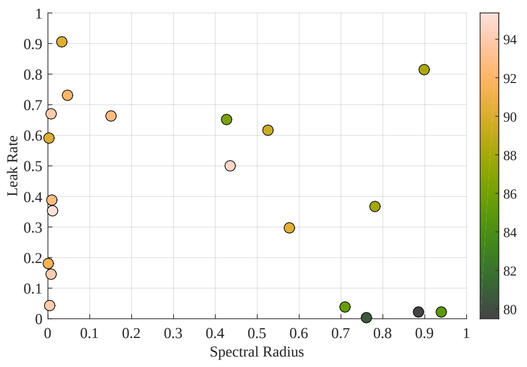
(a) 7 node network - Spectral Radius vs. Input Scaling



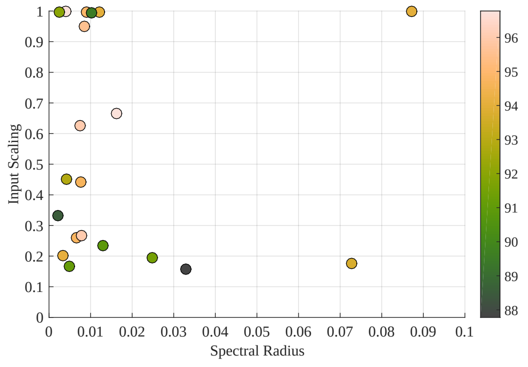
(b) 7 node network - Spectral Radius vs. Leak Rate



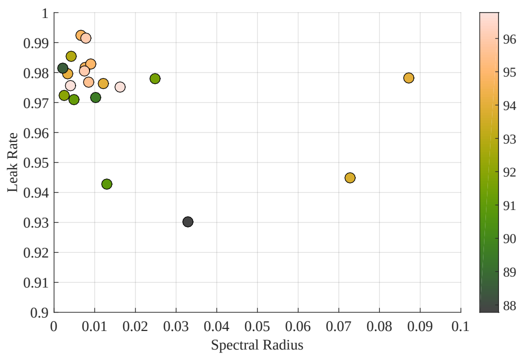
(c) 50 node network - Spectral Radius vs. Input Scaling



(d) 50 node network - Spectral Radius vs. Leak Rate



(e) 7 node (sampled) network - Spectral Radius vs. Input Scaling



(f) 7 node (sampled) network - Spectral Radius vs. Leak Rate

Figure 5.4: Parameters and Iris performances of three evolved Echo State Network sizes.

- Low spectral radius; i.e. does not require a long memory (output depends upon current input) and the reservoir is stable.
- Input scaling around 1; neurons sometimes saturate and become non-linear (see 50-node, Fig. 5.4c). However, there is more variation in dynamics in the 7-node and sampled networks.
- Large variation in leak rates (or close to 1); suggesting the parameter does not significantly effect performance – as expected with other parameters that reduce the reservoirs memory capacity. See Fig. 5.4b, 5.4d and 5.4f.

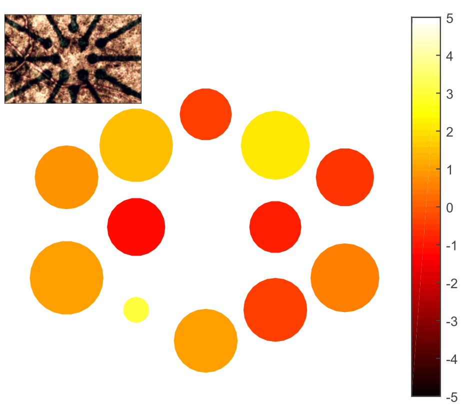
These evolved dynamics align closely with the known *unperturbed* electrical properties of our materials; both semi-stable with modest non-linear *I-V* characteristics. This might suggest why the *in materio* reservoirs perform similarly to *in silico* reservoirs, even after relatively few iterations (typically <50 generations).

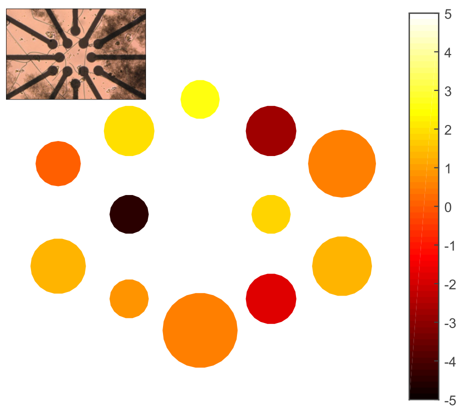
Discovering task-specific parameters for simulated reservoirs – prior to physical implementation – can reduce experiment times. It can also inform the experimenter as to what tasks are reasonable to attempt. With future experiments using new materials, e.g. using materials with dynamic structure, these desirable dynamic traits could "seed" experiments with known configurations that produce such dynamics.

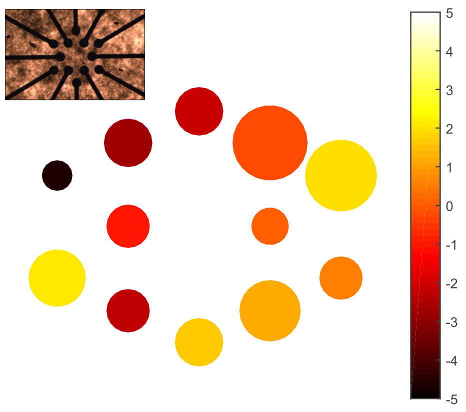
5.3.2 Identifying Substrate Patterns

Mapping the dynamical relationships between different architectural reservoir systems can be enlightening. However, identifying exactly what mechanisms, structure, etc., in the material are being exploited by evolution to produce these dynamical properties is challenging. As discussed in [39], analysing and modelling the nanotube structures and electrical pathways being utilised is difficult. In Fig. 5.5, a simple visualisation is introduced highlighting areas of interesting activity frequently exploited by evolution. The figure displays the electrode arrangement for each material, showing what frequency an electrode is selected as an input (circle size) and what average voltage value is supplied (circle colour) across different evolutionary runs.

The simple visualisation correlates visual carbon nanotube groupings with areas of possible interest. It also highlights regions where there might be weak connectivity


(a) 0.53% PMMA


(b) 0.71% PMMA(1)


(c) 0.71% PMMA(2)

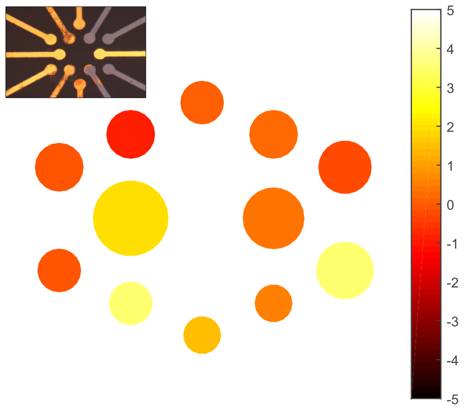

(d) 1% PBMA

Figure 5.5: Visualisation of common evolved electrode patterns.

between electrodes. For example, in Fig. 5.5a, the low frequency at which an electrode is chosen as a stimulus source (shown in circle size) may indicate local isolation in that region of the network/material. This quick visualisation is a useful tool in identifying materials with limited connectivity or homogeneity, as dispersion of the carbon nanotubes is largely stochastic.

5.3.3 Breaking Temporal Dependencies

The material's ability to suppress (or not exhibit) any recurrent dependencies, as seen by the time-independent nature of the task, may suggest certain configurations act more akin to non-temporal kernels or Extreme Learning Machines (ELM) [84], rather than typical temporal reservoirs. To attempt time-independent tasks with simulated reservoirs, reducing the spectral radius and increasing the input scaling parameter minimises any dependencies. Other techniques however have been proposed to “break” the recurrent dependency on previous states and inputs altogether [54]. The interesting point here is that different configurations – on the same materials – have been shown to exploit temporal features on time-dependent tasks and also to vary considerably across different configurations [40, 41]. This suggests tuning of the echo state property is possible through configuration, rather than the material behaving entirely as a non-temporal reservoir/kernel/ELM – this was later proven when characterising the material in Chapter 8.

5.4 Summary

The results presented in this chapter show that by adding the programmable reservoir layer, reservoir computing *in materio* can significantly outperform the original evolution *in materio* implementation. This suggests the RC framework offers improved performance, even across non-temporal tasks, when combined with the evolution *in materio* technique.

This increase in performance, however, might be attributed to the new measurement and stimulation technique. For example, performance might increase as a result of: (i) stimulating the material with variations of the input causes interesting interactions not present in the EiM technique; (ii) a conductive network might not be present across all electrodes and instead several networks may exist across the array, therefore additional inputs-outputs grant access to each of these networks; (iii) combining the outputs and weighting their importance allows training to exploit the whole material rather than exploiting a single area around a particular electrode (relating closely to (ii)). Each of these hypothesised explanations still requires further exploration.

At the end of the chapter it was shown (in simulation) that sampling only a few reservoir states, thus having fewer trainable states, can affect performance. We also characterised the dynamical behaviours of both physical and simulated reservoirs through global parameters. From this, it was explained why smaller simulated networks perform similarly to the carbon nanotubes, i.e. due to matching non-linearity and memory requirements, and fewer observable states.

In the next chapter, a new hardware platform is outlined. The platform attempts to address limitations discussed in previous chapters by adding more inputs and outputs (electrodes), greater flexibility, and measurement strategies with greater precision.

Chapter 6

An Extended Hardware Platform

6.1 Input-Output Complexity Problem

The most common form of evolution *in materio* uses a micro-electrode to interface with the substrate. This puts a hard limitation on the number of inputs and outputs available. However, the direct encoding of the computational task, e.g. a task with five inputs requires five electrodes, adds another limitation to the system as well. This leads to a fundamental limit on the complexity of tasks that can be performed, and explains why much of the work to-date has been bound to simple tasks.

For such problems, the reservoir computing framework has some desirable features. The framework itself adds layers of abstraction to compensate for limitations in hardware. For example, the reservoir readout layer creates *virtual* task outputs that combine activity from across the substrate, reusing and combining electrode outputs, instead of assigning individual task outputs to one spatial location. In comparison, EiM assumes activity at one electrode is sufficient to fulfil the task.

Despite advantages, the framework does not fully resolve the scaling and task complexity problem either. New obstacles also emerge. For example, how to connect to and encode the task inputs for individual electrodes; is an input weight matrix sufficient? is a greater complexification of the input needed? how can a large input space, say an image, be encoded into few electrodes?

Another dilemma is discussed in Chapter 5, where it was shown that sub-sampling of the reservoir states can significantly affect performance. If we postulate that the number

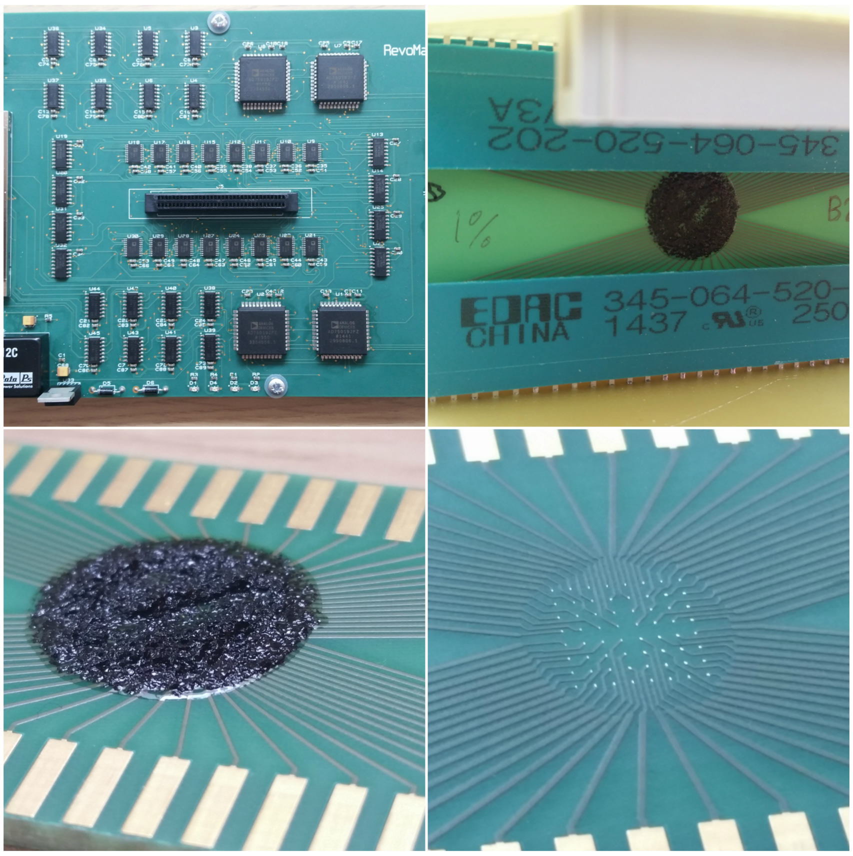


Figure 6.1: Hardware Reservoir System. Micro-electrode housing, routing switch board and CNT/polymer deposited onto PCB electrode array.

of unique local states are vast in a physical substrate but we can only observe a select few, the number of electrodes available still becomes a pertinent issue.

To improve our understanding of the scaling problem requires observations at different scales. So far, experiments have been largely limited to a maximum of 16-electrodes. For this reason (and others stated in section 4.5.5) a new hardware platform is presented.

6.2 Extended Hardware Platform

The design process for the new extended hardware platform consisted of two iterations. In the first iteration, several anomalies emerged, resulting in very good but questionable task performances. These were then analysed and compensated for in the second iteration.

Overall, the basic design features of both iterations are as follows. A total of 64 electrodes are available to use, with a maximum of 32 inputs that can be used in parallel. These design limits were imposed by the analogue output (AO) National Instruments Data Acquisition (DAQ) Card, accepting only 32 analogue outputs.

To extend the hardware system to 64 electrodes required the design of: a new routing system (previously a single 16×16 cross-point switch), a new micro-electrode design, a housing for the substrate, and the fabrication and depositing of carbon nanotube composites. The final outcome of each is shown in Fig. 6.1[1].

6.2.1 Signal Routing

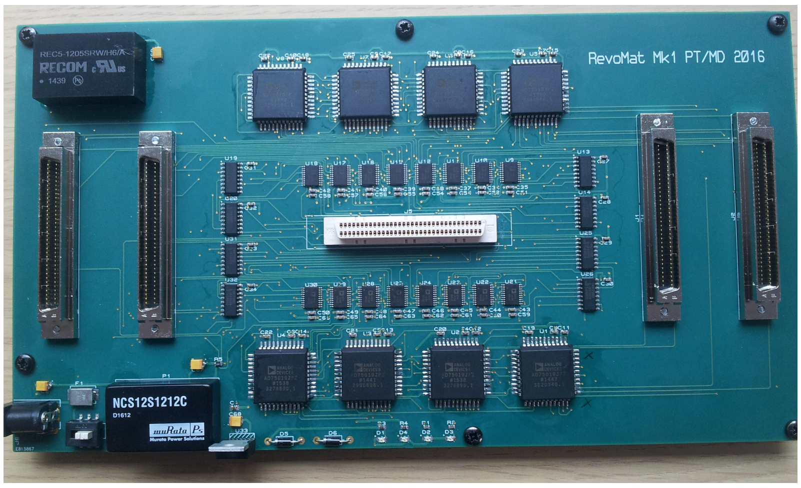
Due to the limited number of analogue outputs, routing a voltage signal from the AO DAQ card to any of the 64 electrodes requires some intermediate switches. For this purpose, an additional routing switch-board was designed. In an ideal scenario, the number of analogue outputs and inputs would be equal, removing this intermediate step.

Details of the routing switch-board, auxiliary equipment and the DAQ cards are as follows. Two National Instruments DAQ cards perform measurements and output analogue voltages; a PCI-6225 (16-Bit, 250 KS/s, with 80 analogue inputs), and PCI-6723 (13-Bit, 800KS/s, with 32 analogue outputs). Both cards communicate to a desktop PC through a session-based interface in MATLAB. The PCI-6723 supplies 8 digital I/O lines to the custom routing board to program on-board switches and synchronise the cards.

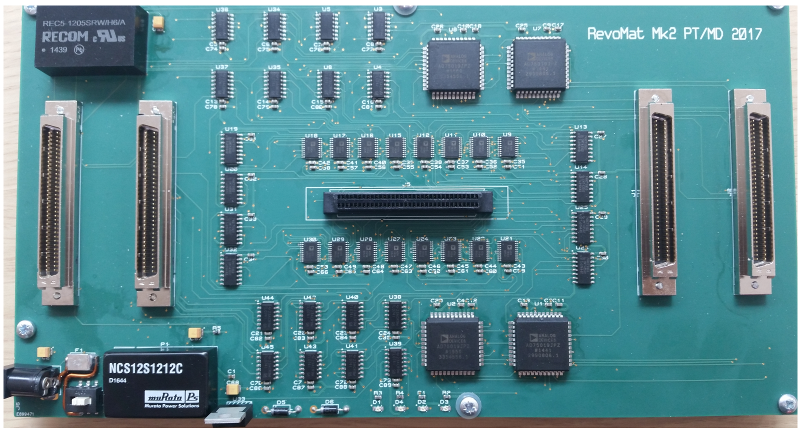
In the first design iteration, an array of eight 16×16 analogue cross-point switches were used to connect the 32 output channels on the PCI-6723 to the substrates input-electrodes. In the second iteration this was reduced to four switches, thanks to a better routing strategy.

In the second iteration, the 32 channels were divided among four switches (A, B, C, D), with channels 1–16 feeding into A and B, and channels 17–32 into C and D. Switches A and B cover the first 16 channels that map to the first 32 electrodes, and C and D cover the next 16 channels mapping to the last 32 electrodes. This maps the 32 output channels to all 64 possible electrodes. As a minimum requirement to stimulate the substrate, one output DAQ channel/input-electrode is always required, which is forced in software.

[1]The co-design of the routing board, micro-electrodes and housing was carried out with Pete Turner from the Electronic Engineering department.



(a) Iteration 1

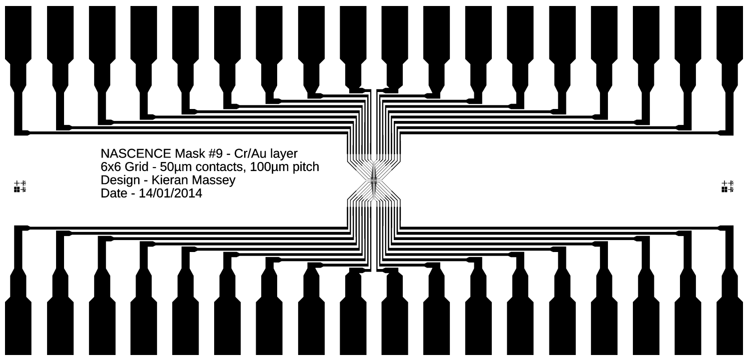


(b) Iteration 2

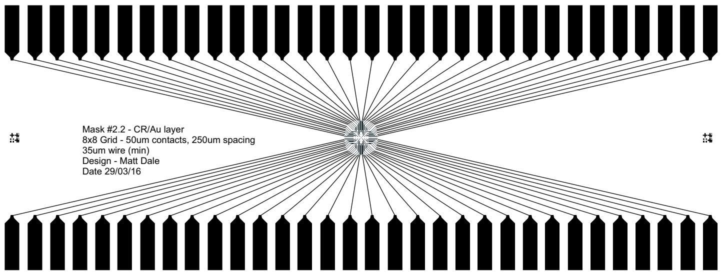
Figure 6.2: Pictures of both routing boards.

Beyond the cross-point switches, independently selectable single-pole, double-throw (SPDT) switches are tethered to each electrode to allow the computer to dynamically map each electrode to an input, or output. These were added to simplify the programming step, to reduce crosstalk and reduce the number of crosspoint switches required. In previous chapters using the smaller system, both inputs and outputs fed through the crosspoint switches increasing the likelihood of crosstalk and unnecessarily complicating the configuration process. To control the SPDT switches, 8-bit shift registers were used, programmed directly by the digital I/O lines of the PCI-6723 card.

The final boards of both iterations are shown in Fig. 6.2. Early ideas and the full designs of each board are also given in the appendices, see Appendix A.1.


NASCENCE Mask #9 - Cr/Au layer
6x6 Grid - 50µm contacts, 100µm pitch
Design - Kieran Massey
Date - 14/01/2014

(a) NASCENCE project (unused) 32-electrode design



(b) 64-electrode design

Figure 6.3: Micro-electrode array designs. a) example from NASCENCE project. b) new 64-electrode design.

6.2.2 Micro-electrode Arrays

In previous chapters, the number of electrodes available were small, typically 12 to 16 electrodes. The micro-electrode arrays used were designed and fabricated by Durham University members of the NASCENCE project. The fabrication process of those micro-electrode arrays used etch-back photolithography, placing chromium/gold-contact electrodes (40 to 50 µm contact diameter and 100 to 150 µm contact spacing) on glass slides, either arranged in a circle or grid array.

The initial plan for the new hardware platform was to design and fabricate similar micro-electrode arrays on glass slides; see Fig. 6.3 and 64-electrode designs/masks in Appendix A.3. However, accessing suitable equipment and expertise to complete the plans proved difficult.

Instead, a new fabrication process was investigated where the micro-electrode array would be etched onto a specially designed printed circuit board (PCB). In the process, the electrodes and pads are deposited onto a FR-4 PCB using a chemical process that places Nickel and then a layer of Gold. A photo-imageable solder mask is then applied to cover all excess electrode surfaces, exposing only the designated contact points. After fabrication, basic tests were conducted measuring the effect of moisture absorption. This effect was considered negligible, and the designs were kept.

The new PCB electrode designs make fabrication of the MEAs simple, quick and cost-effective in comparison to the previous glass photo-lithography technique. However, due to manufacturing constraints, the electrode array requires larger contact sizes of 100μm and spacings of 600μm between contacts compared to previous designs. According to Massey [132], closer spaced electrodes will exhibit larger currents and resistance will generally scale approximately with electrode separation. How the current contact sizes and spacings affect the new set-up still requires further investigation.

6.2.3 Fabrication Process of Nanotube Composites

The composites used with the new hardware platform comprise of single-walled carbon nanotubes (CHASM ADVANDCED MATERIALS, Sigma Aldrich) created using catalytic chemical vapor deposition (CVD). The nanotubes have an average diameter of 0.78 nm and a median length of 1μm. The sorted nanotubes comprise $\geq 95\%$ carbon basis ($\geq 99\%$ as carbon nanotubes) with approximately 95% semiconducting, and approximately 41% of those tubes being (6,5) chirality.

The mixing materials are poly-butly methacrylate (PBMA) (Sigma Aldrich, Mw $\approx$ 337,000, powder) and Methoxybenzene/Anisole (Sigma Aldrich, analytical reagent grade). These are added, entrapping the nanotubes and help to form solid network structures.

The process is simple with no top-down design involved, allowing the nanotube networks to randomly form conducting pathways and insulating polymer regions. This results in a non-uniform dispersion and some nanotube bundling as a result of weak short-range electrostatic attractive forces.

The mixing and depositing details of the composites are as follows. The mixtures contain: 5g Anisole, 1g PBMA, and 50mg carbon nanotubes in dried powder form, resulting in $\approx$1% (w.r.t. weight) nanotube mixture. The solution is left to dissolve for 24 hours, converting the powdered PBMA into a viscous liquid. Once dissolved, the solution is placed in an ultra sonic bath for 20 minutes at full power to aid dispersion of the

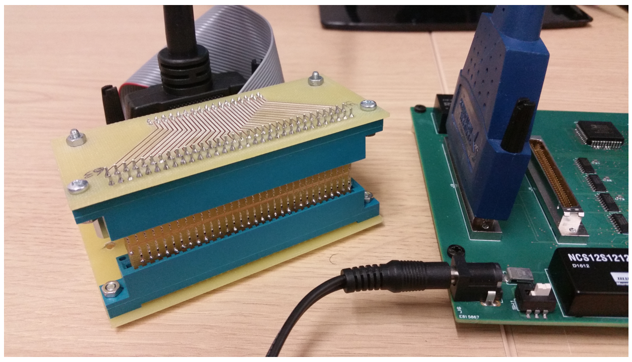


Figure 6.4: Housing for PCB; connecting substrate to routing board to PC.

nanotubes. Then approximately 30μl of the mixture is dispensed onto a pre-heated PCB and left to dry on a hotplate for 30 minutes at 85°C. To reduce stress in the cooling stage, the hotplate is turned off and the PCB is left to cool naturally for two hours.

Each deposited material is unique, however properties like conductivity tend to be consistent when using the same percentage of carbon nanotubes. Materials from the NASCENCE project with similar compositions were shown to produce non-linear behaviour seen in the current (I) versus voltage (V) measurements, depending on mixture ratios and film thickness [199]. Examples of *I–V* curves for different nanotube mixtures are given in Appendix A.4.

6.2.4 Substrate Housing

To make interfacing with the substrate as simple as possible two PCB edge connectors were used. These 32-contact connectors (EDAC 345-064-520-202) are standardised and can be purchased off-the-shelf. The PCB micro-electrode array itself was designed to fit the 2.54mm pitch between contacts, giving the platform its *plug-and-play* simplicity.

Each edge connector is placed on a prototype PCB board. These separate boards are then connected via a ribbon cable, then connected to a d-type 64-pin shielded cable that communicates with the routing board. A picture of the housing is given in Fig. 6.4.

6.3 Measurements and Stability

Considering previous work on solution stability (see section 4.5.3), the assumption that the materials are *always* within some stability bound when configured appears inaccurate. Despite many configurations being stable, it was frequently seen that for some input-output mappings large deviations were present when repeatedly applying the same inputs. Therefore, the *precision* of the measuring system appeared to be low. The cause of this is referred to here as solution or configuration *drift*. Such configurations are considered detrimental, unstable, reduce repeatability, complexify the evolutionary search and reduce confidence in measured performances.

To measure drift with the new system, multiple evaluations are carried out every time the substrate is stimulated. Drift is measured separately for each electrode as the difference (e.g. Root Mean Square Error (RMSE)) between evaluations i.

The resulting states of each evaluation are $x_i = \mathbb{R}^{N_u \times N_{elec}}$, where N_u = number of input samples and N_{elec} = number of electrodes. All states are collected and stored in the matrix X for the drift calculation, as follows:

$$X = \mathbb{R}^{N_{eval} \times N_u \times N_{elec}} = \begin{bmatrix} x_1, & x_2, & \dots & ,x_N \end{bmatrix} \tag{6.1}$$

The drift for each electrode is calculated as the median error across all combinatorial pairs $\binom{N}{i}$. This value is then stored. In the next phase, using the drift values a decision is made whether to manipulate the final training states, or not. First, the collected states are averaged to give X', the median of N_{eval} evaluations, $X' = median(X) = \mathbb{R}^{N_u \times N_{elec}}$. Then, X' is processed to remove high drift electrodes, setting electrode states ($X'[:,i]$) to zero when electrode variance is high (above some user-defined threshold), or when an electrode is assigned as an input location.

As part of the training process, zero weights are assigned to unwanted electrodes; tricking the training process into seeing no observable output. This results in unwanted noisy electrodes being ignored for future evaluations, such as test and validation data. The whole drift measurement is therefore only carried out at the training stage, and inherently compensated for in other evaluations via output weights.

The above process removes unstable outputs, however it does not help prevent unstable solutions occurring during the search process. Actively reducing these deviations though evolutionary selection, or output smoothing, can result in improved performance. In preliminary experiments, where drift information was provided, this was shown to be true. However, evolutionary selection in many cases was also seen to prioritise stable solutions without providing information about drift.

Sources of drift and noise can be both internal and external. An example of two potential sources are: i) after each evaluation the material is “reset” by grounding all electrodes, however, some charge may remain between evaluations in localised sub-networks; ii) the material may physically change, e.g. nanotubes can move, “burn-out”, or conductivity may degrade over-time, thereby changing the material’s phase space, naturally resulting in non-static solutions and performance drift over-time.

In general, any practitioner of physical reservoir computing with non-deterministic substrates should be aware of, and take appropriate measures to compensate for, effects such as solution/configuration drift. Furthermore, in terms of any future applications, any *drift* and change in performance with time should be reduced or compensated for, especially if the system is designed to be a robust alternative computing solution.

6.4 Genetic Representation

For the platform, a new genetic representation was designed to take into account hardware limitations, and make use of extended capabilities not possible in the previous hardware system, for example, having both evolvable input weights and control voltages available at the same time (see Chapter 4).

The genetic representation of each reservoir, i.e. encoding the configuration of the substrate, is stored in the $G_{Config} \times G_{Inputs}$ matrix G, where rows G_{Config} define different configuration types and columns G_{Inputs} are equal to the total number of available inputs. An example configuration G is shown in Fig. 6.1. Here, $G_{Config} = 6$ and individual genes $G_{i,j}$ can be flipped to active or non-active depending on other coupled genes.

In the example, genes $G_{(1,j)}$ represent the electrode under manipulation. These range from 1 to 64 and cannot be duplicated. The restriction of $G_{Inputs} = 32$ implies no more than 32 inputs can be active. In fact, the matrix G is only used to define inputs as any

Gene	1	2	3	4	$\cdots$	$\cdots$	30	31	32
Electrode	22	13	4	44	$\cdots$	$\cdots$	64	52	1
Input(Yes=1)	0	1	0	0	$\cdots$	$\cdots$	1	0	0
Weight(1)/Control(0)	0	1	1	0	$\cdots$	$\cdots$	1	1	0
Weight (V)	3.12	1.23	-0.12	4.12	$\cdots$	$\cdots$	2.36	1.78	0
Control (V)	0.15	-0.24	2.31	0	$\cdots$	$\cdots$	-3.21	4.9	0
Leak rate	0.56	0	0	0	$\cdots$	$\cdots$	0	0	0

Table 6.1: Encoding G for Reservoir and Substrate Configuration.

electrode not defined in G is automatically an output by default.

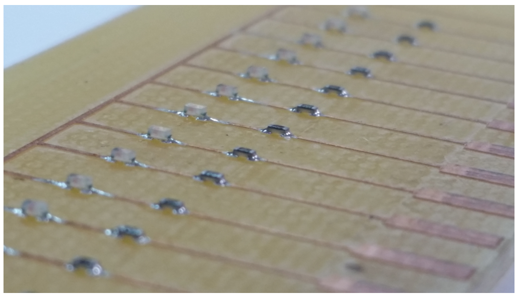
The genes $G_{(2,:)}$ determine whether the electrodes $G_{(1,:)}$ are inputs. If $G_{(2,j)} = 1$, electrode $G_{(1,j)}$ becomes an active input. What type of input the electrode becomes depends on row $G_{(3,:)}$. If $G_{(3,j)} = 1$, the electrode is assigned as a weighted task input. Otherwise, it is set as a control input, i.e. static voltage. This gene is only active when $G_{(2,j)} = 1$, otherwise it is redundant.

Depending on whether electrode $G_{(1,j)}$ is an active input, and a weighted task input or control, determines whether the next two rows ($G_{(4,j)}$ and $G_{(5,j)}$) are active. If $G_{(3,j)} = 1$, the weight/gain value $G_{(4,j)}$ is applied to the task input in the digital domain. Or, if $G_{(3,j)} = 0$ the value in $G_{(5,j)}$ is used as the voltage value for the static control signal.

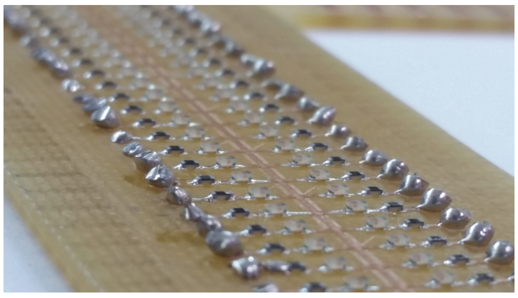
The last row $G_{(6,:)}$, holds leak rate values. At the moment, a single global leak rate is applied, however individual leak rates for each electrode are possible.

In future work, the input weight matrix W_{in}, different control types (not only static voltages), and even output weights W_{out} could be added to the genetic representation.

An advantage of the current encoding G is that it is considerably smaller than other encodings (e.g. a digital echo state network), requiring significantly less storage. This very basic advantage can be easily overlooked. However, this fundamental feature allows vast numbers of configurations/abstract reservoirs to be stored and updated in conventional memory, simply implemented via the turn of a switch.


(a) 16-LED

(b) 16-LED in housing


(c) 64-LED

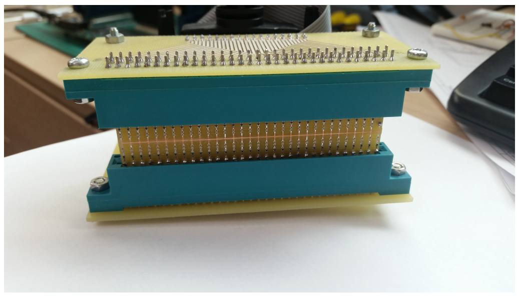

(d) 64-LED in housing

Figure 6.5: Two LED designs printed on PCBs.

6.5 Isolating the Substrate

As part of the testing phase of the hardware, small PCBs with light emitting diodes (LEDs) were designed and built to visually test the routing board (see Fig. 6.5 and Appendix A.5). Out of curiosity, these LED arrays were also evolved on previous reservoir computing tasks, the results of which were surprising.

Beforehand, it was assumed the LED arrays would perform poorly as limited, or no network connectivity between LEDs is present. Therefore, sufficient memory and network dynamics would not be available to solve such tasks. However, unexpected results were produced: For the NARMA-10 task, there was $\sim 30\%$ increase in performance, from an NRMSE ≈ 0.71 (in Chapter 3) to an NRMSE ≈ 0.51. Remarkably, this echoed similar performances to other, much larger physical reservoirs such as the 100-node optoelectronic reservoir (NRMSE ≈ 0.5) in [153]. For the wave generator problem in Chapter 3, an 92% increase in performance was achieved. And finally, for the Santa Fe laser prediction task a 25% increase was evolved. These striking performance gains suggested properties such as memory were being exploited from somewhere else in the hardware.

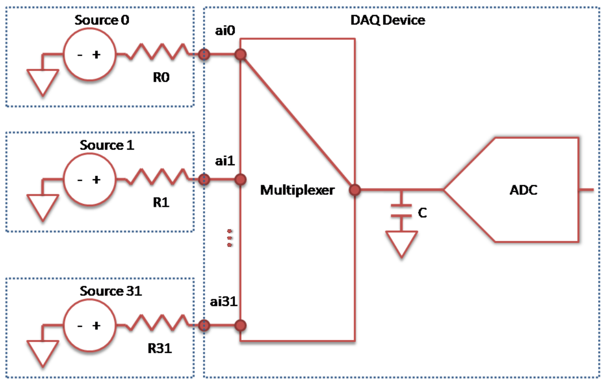


Figure 6.6: Multiplexed analogue-to-digital converter (ADC) from National Instruments documentation on "ghosting".

When further analysing the connectivity between the DAQ cards and the routing board (using voltage probes), it was discovered that a major contributing factor to the memory phenomenon occurred during the measurement process, in the analogue input DAQ card.

The DAQ card transforms each analogue input-channel into a digital signal sequentially through a multiplexed analogue-to-digital (ADC) converter. This multiplexed process requires the dissipation of a capacitor between the ADC and the multiplexor (see Fig. 6.6). If charge remaining in the capacitor is not released between samples, information from previous samples can persist and reflect across channels. This is known as "ghosting". Solutions to removing ghosting can be found on the National Instruments web pages[2].

To remove ghosting, voltage buffers/followers were added to the next iteration of the routing board. The analogue voltage buffers chosen were implemented using op-amps (TL047) configured to unity gain.

Applying buffers removes measurement errors when any configuration forms an impedance on an output electrode greater than the input impedance of the measurement device. Effectively, this sets floating electrodes to 0V. In some rare cases, this can result in no usable output signals from the substrate.

As highlighted in this section, it is critical not to overlook that the whole interface will

[2]National Instruments web page on eliminating ghosting: `https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019KzzSAE`, Accessed: 22/08/2018

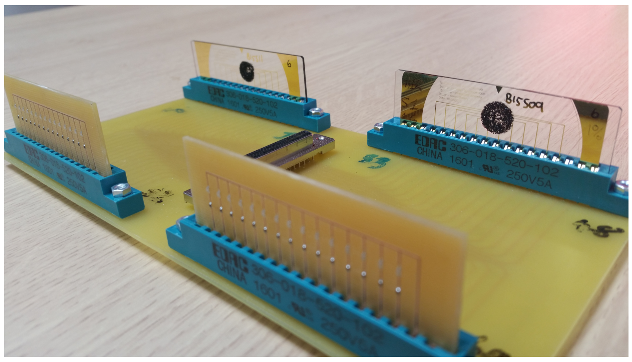


Figure 6.7: Multi-substrate Housing.

in some way affect the perceived computation. In other words, the reservoir is not only the substrate but the interfacing equipment too.

It is imperative with any physical reservoir system to make sure the substrate being exploited is sufficiently isolated from the rest of the system. In equation 2.16, this is represented by the Ω function, as the observation process itself may perform some computation.

In general, this phenomena unique to physical substrates may be viewed as a trade-off problem. If the substrate is fixed permanently to the measuring equipment, the proportion of computation performed by the interface may be considered less crucial. However, if the substrate is to be trained on one device and then applied on another, the substrate under-test should be the main, if not the only, contributor.

6.6 Multi-Substrate Housing

During the hardware upgrade, a number of minor decisions were made to allow expansion into multi-substrate computing. Although not fully exploited yet, basic steps have been made to test the concept. The first step has been to fabricate a housing for multiple substrates.

The multi-array housing allows parallel substrate evolution; evolving multiple independent substrates at the same time. This basic prototype PCB housing is shown in Fig. 6.7, where four 16-electrode connectors are combined and connected to the routing board in the same fashion as the 64-electrode housing.

In a few brief experiments, the same genetic representation G has been used; multichromosome representations, however, might be a better solution. The states from each substrate were used to train *one* reservoir that is effectively made from smaller independent reservoirs, e.g. equivalent to an ensemble of reservoirs. So far, early results have been encouraging, however, more work is required.

In future work, such a reservoir may be formed from substrates with very different characteristics, leading to a wider dynamical range than a single reservoir can possess. This work could also entail different methods to interconnect each substrate. However, at this stage, the benefits of multi-reservoir/multi-substrate computers are still speculative and require further investigation both in software and hardware.

6.7 Summary

In this chapter a new hardware platform with 64 electrodes is outlined. This required new materials, micro-electrode arrays, genetic representations, and measurement strategies to be created.

During the design process, a significant problem was identified. Evolution was found to be exploiting the measurement system as well as the material, leading to unusually high performances. The phenomenon highlighted a general cause for concern when defining the boundaries of any physical computing system, i.e. is the computation isolated to the substrate or is the observation equipment contributing, if the latter, what is an acceptable amount.

In the final section, it was shown how the platform can be extended to measure multi-substrate reservoir systems, providing new avenues for future work.

In the next chapter, the concept of multi-reservoir systems is explored. A new evolvable architecture is proposed allowing evolution to design hierarchical structures for specific tasks.

Chapter 7

Hierarchical and Modular Reservoir Computers

7.1 Limitations of Reservoirs

All reservoirs take advantage of a fixed and typically random topology, often requiring a careful selection of global parameters to control reservoir activity in order to exhibit desirable dynamical properties. ESNs are an exemplar of this tuning problem.

In general, ESNs consist of random fixed recurrently connected analogue neurons. They are powerful and efficient systems often featuring state-of-the-art performance across many domains, with applications in small embedded systems, sensors and robot controllers [8, 172]. However, as with any reservoir computing system, it has limitations and caveats. Good reservoir parameter selection and weight initiation are essential to produce high performing ESNs.

Reservoirs and ESNs also possess another fundamental limitation; an inability to model multiple time scales and levels of abstraction. This is because a single reservoir can only represent information at a single scale due to its network coupling and dynamics [120].

In the literature, techniques have been proposed to solve both the reservoir parameter selection and the multi-scale problem using varying degrees of complexity. For example, reservoir optimisation is a popular research area using techniques such as particle swarm optimisation [12, 167], Bayesian optimisation [204], gradient-based information [205], and evolutionary algorithms [31, 56, 93, 133, 156]. To solve the multi-scale problem, different architectures and constraints have been proposed, varying from highly structured hierarchies to clustering approaches that transition between heterogeneous and homogeneous structures. These include the dynamical feature discoverer (DFD) model, ensembles of ESNs, decoupled hierarchies of ESNs, "deep" multi-layered structures, and hierarchical clustering [61, 89, 92, 166, 203].

The general problem of finding optimal parameters for any reservoir system can be challenging, and as model complexity increases so does the difficulty in selecting suitable parameters. In the literature there are a few examples combining optimisation and modularity [124, 125, 156, 207] in an attempt to reduce the difficulty in designing topologies, internal weights and parameters. However, evolving hierarchical structure, network connectivity and global reservoir parameters together – essentially all parameters at the same time – has not yet been attempted. This may have been unappealing in the past as the genetic representation can explode in complexity and significantly increase the search space. However, if implemented well, it could potentially add much more evolvabilty, leading to greater single-task and multi-task performances.

In this chapter a new evolvable architecture is proposed and investigated called the *Reservoir-of-Reservoirs* architecture. Its primary design is to expand the capability of hardware-based reservoir systems, enabling the future possibility of multi-substrate reservoir systems.

Hardware-based reservoirs feature the same limitations as virtual reservoirs, as well as some extra substrate specific limitations. For example, reservoir size and internal parameter tuning may be restricted. The restriction on reservoir size, e.g. the number of physical inputs and outputs available, can severely impact the task potential of such systems. Therefore, combining many smaller reservoirs and evolving hierarchies could lead to more complex networks that would otherwise be too problematic to realise in hardware. This idea has been briefly discussed [39, 41] for small unconventional substrates, and demonstrated physically with hierarchical compositions of memristive networks [22].

7.2 Hierarchical Reservoir Computers

Hierarchical learning systems provide intermediate levels to extract and specialise on features, feeding from one layer into another, creating new abstract features from subsequent ones. In terms of recurrent learning systems, this layering separation allows multiple independent temporal and spatial scales to be learned [89].

Hierarchies are essential in biological neural networks and have become popular with deep learning systems. The neocortex is a prime example exploiting structural hierarchy featuring six layers and billions of neurons communicating through different layered channels with specialised connectivity [149]. Each layer contains different neuronal shapes, sizes and densities, with different organisational structure; this heterogeneity is often undervalued in artificial neural networks.

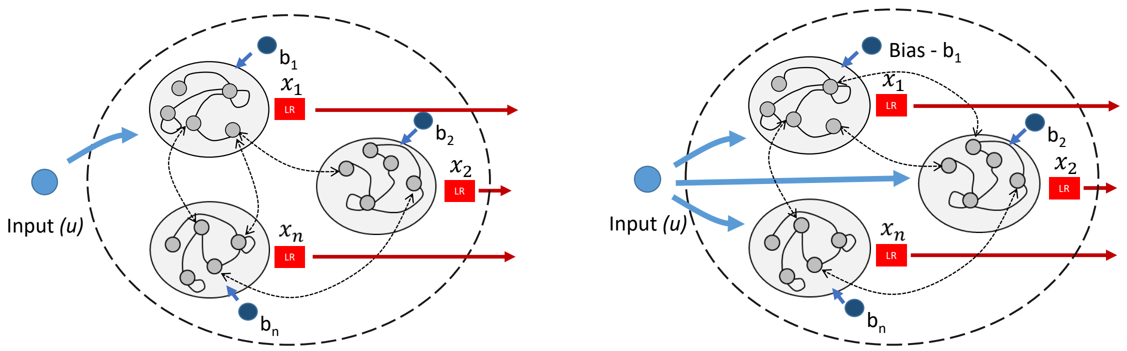
In the reservoir computing literature, different hierarchies of ESNs have been proposed, exploiting the ESN structure and training separation that offers simple hierarchical organisation and a fast training mechanism. Some of those proposed include: decoupled echo state networks (DESN) [203], scale-free highly clustered ESNs (SHESN) [48], hierarchically clustered ESNs (HESN) [92], and Deep ESNs [61, 62, 124, 125].

7.2.1 Reservoir of Reservoirs

The Reservoir-of-Reservoirs (RoR) architecture, recently proposed in [38], comprises a reservoir containing smaller reservoir networks. The architecture utilises the classic ESN training segmentation of three layers (input, hidden and readout layer) where only the final readout layer is trained. The hidden layer, referred to as the *master* reservoir, connects every *sub*-reservoir network of neurons to each other, as shown in Fig. 7.1.

Two versions of this architecture are proposed: *RoR* (Fig. 7.1a) and *RoR-IA* (Fig. 7.1b). The first receives the task input at only one sub-reservoir, and the latter features inputs to all sub-reservoirs.

The difference between the two is a restriction in information flow, i.e., the effect input data at time t has on the current sub-reservoir state. This also implies that some sub-reservoir states rely heavily on residual information, i.e. a sub-reservoir that does not receive the input directly receives a transformed version of the original input data. This



(a) RoR (b) RoR-IA

Figure 7.1: Reservoir-of-Reservoirs (RoR) architecture: a) only one sub-reservoir is supplied with the input u, b) all sub-reservoirs have access to the input u. Each sub-reservoir has independent parameters, such as biases b_i and leak rate α_i. All states x_i are used for readout training.

can potentially lead to multi-scale projection and abstraction if tuned correctly, but could also lead to degenerative information, e.g. destructive transformations or randomisation of the data, that could persist in the system.

The RoR architecture is described, like classic ESNs, by a series of weight matrices. Each sub-reservoir consists of an independent input matrix $W_{in}^r \in \mathbb{R}^{I \times N_r}$ and an internal matrix $W^r \in \mathbb{R}^{N_r \times N_r}$ where N_r is total number of nodes in sub-reservoir r. Every sub-reservoir has local network parameters, such as input scaling $a(W^r)$ and scaling of W^r that can be manipulated be evolution.

To form the master reservoir each sub-reservoir connects to every other sub-reservoir via the sparse inter-reservoir matrix $\bar{W}$ of matrices, see Eqn. (7.1). This weight matrix represents internal weights of each sub-reservoir ($W_r = W_{ij}$, when i and j are equal to r) and the connections between each sub-reservoir. The weights connecting each sub-reservoir are set to feed directly to neurons/nodes within other sub-reservoirs ($W_{ij} \in \mathbb{R}^{W_i \times W_j}$), bypassing the task input layer of each sub-reservoir.

$$\bar{W} = \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,r} \\ W_{2,1} & W_{2,2} & \dots & W_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ W_{r,1} & W_{r,2} & \dots & W_{r,r} \end{bmatrix} \tag{7.1}$$

The state update at time t for each sub-reservoir $x_i(t) \in i = 1, \dots, r$ is defined in Eqn. (7.2). Separate leak rate filters are added to the output of each sub-reservoir, as shown by the red (LR) boxes in Fig. 7.1, which can be adjusted using individual leak rate parameters α_i. The weighted effect previous states of other sub-reservoirs have on sub-reservoir r is defined in Eqn. (7.3).

$$x_i(t) = (1 - \alpha_i)x_i(t-1) + \alpha_i f\left(u(t)W_{in}^i + S_i(t)\right) \tag{7.2}$$

$$S_i(t) = \sum_{j=1}^{r} \bar{W}_{ij} x_i(t-1) \tag{7.3}$$

Applying the classic ESN approach, the activation states of *all* neurons in the network are collected to perform training, resulting in the prediction $y(t)$ defined as:

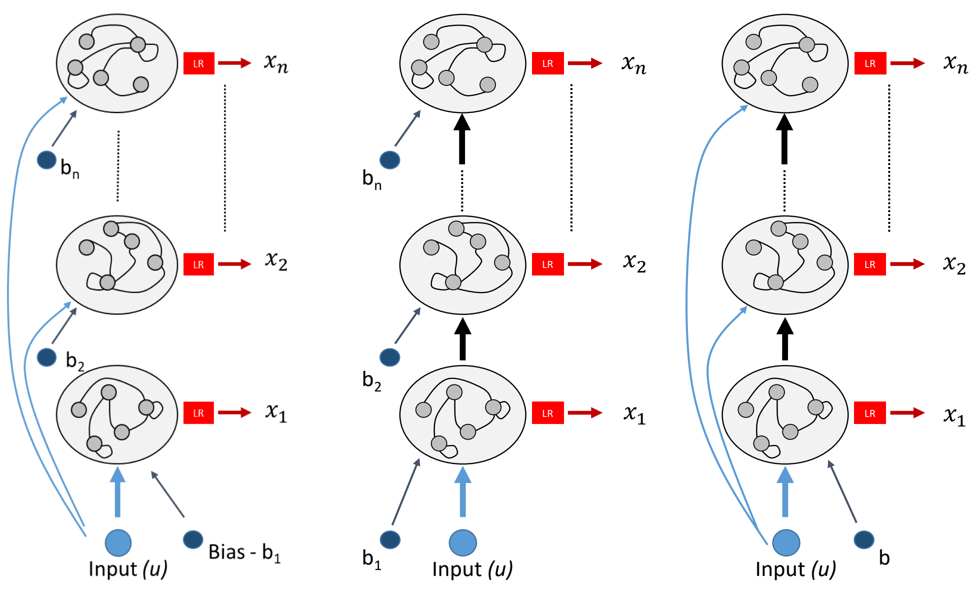
$$y(t) = W_{out}[1\ u(t)\ x_1(t)\ x_2(t) \dots x_r(t)]^T \tag{7.4}$$

At creation, every sub-reservoir's internal matrix is initiated as a sparse network with 10% connectivity. In the literature, the sparseness of W_r is typically considered more beneficial to update speed rather than performance [117]. Every other weight matrix connecting sub-reservoir i and j is initiated as a sparse network with 1% connectivity. The matrices W_{ij} and W_{ji} in $\bar{W}$ are identical, representing one set of bidirectional weights. An additional reservoir parameter $\phi(W_{ij})$ determining the scaling of each W_{ij} is also set.

Despite each matrix being initialised as sparse, weight mutation is free to adapt connectivity over time. Mutation can also create or break recurrent loops between sub-reservoirs, increase homogeneity and remove or add hierarchy. Crucially, the necessity to utilise hierarchy, recurrence or homogeneity depends on the task being evolved for.

7.3 Comparison Architectures

To evaluate the RoR architecture, three additional architectures are implemented based on examples from the literature. The first is an ensemble of n independent ESNs shown in



(a) Ensemble (b) DeepESN (c) DeepESN-IA

Figure 7.2: Hierarchical architectures: a) Ensemble of independent sub-reservoirs, b) Pipeline of sub-reservoirs with input u only at first sub-reservoir, c) Pipeline with input u supplied to every sub-reservoir. All have independent bias terms b_i and use neuron states x_n (after leak filter) for readout training.

Fig. 7.2a, inspired by [166]. The architecture divides the task among multiple networks and the combined reservoir states produce the desired output. For every network in the ensemble there are again independent reservoir parameters, such as the bias term (b_i all being the same) and leak rates α_i. The state update equation for the ensemble is given by:

$$x_i(t) = (1-\alpha_i)x_i(t-1) + \alpha_i f\left([b_i\ u(t)]W_i^{in} + W_{ii}x_i(t-1)\right) \tag{7.5}$$

The next two architectures are based on a multi-layered pipeline ESN, referred to here as *DeepESN*, inspired by [62]. There are two variations of the DeepESN, as shown in Fig. 7.2b and Fig. 7.2c. For the standard DeepESN the input enters only the first layer, whereas for the DeepESN-IA the input is provided to all layers. The state update equations for the DeepESN and DeepESN-IA are given in Eqns. (7.6), (7.7) and (7.8). For the DeepESN, the input matrix is set as $W_i^{in} \in \mathbb{R}^{I\times N_i}$, if $i = 1$, and $W_i^{in} \in \mathbb{R}^{N_{i-1}\times N_i}$, if $i \neq 1$.

$$\hat{x}_i(t) = \begin{cases} f\left([b_i\ u(t)]W_i^{in} + W_{ii}x_i(t-1)\right) & \text{if } i = 1 \\ f\left([b_i\ x_{i-1}(t)]W_i^{in} + W_{ii}x_i(t-1)\right) & \text{if } i \neq 1 \end{cases} \tag{7.6}$$

$$\hat{x}_i(t) = \begin{cases} f\left([b_i \; u(t)]W_i^{in} + W_{ii}x_i(t-1)\right) & \text{if } i = 1 \\ f\left([u(t) \; x_{i-1}(t)]W_i^{in} + W_{ii}x_i(t-1)\right) & \text{if } i \neq 1 \end{cases} \tag{7.7}$$

$$x_i(t) = (1-\alpha_i)x_i(t-1) + \alpha_i\hat{x}_i(t) \tag{7.8}$$

7.4 Optimisation: Microbial GA

Manually selecting reservoir parameters can be a cumbersome task. Difficulty and effort increase with hierarchical architectures as the number of parameters and interactions between units increase. However, the reservoir structure lends itself well to evolutionary optimisation as all reservoir layers and parameters are adaptable. Examples of evolutionary methods applied to ESNs include: evolving input scaling, spectral radius scaling and leak rate parameters, network sizes, and topologies with repeating patterns and geometric regularities [31, 56, 133, 156]. Evolutionary optimisation can also be used to apply ESNs to unsupervised problems when no teacher input-output data is available, as demonstrated in [93].

To optimise the proposed architectures a *steady-state* genetic algorithm (GA) called the *Microbial* GA (M-GA) [76] is used. The steady-state algorithm allows individuals to survive across many generations, provides elitism for free, and offers a simple mechanism for selection, recombination and mutation. It also has the advantage of being well suited to distributed and asynchronous applications which can be useful for accelerating the evolutionary process [76].

An example of the microbial GA implementation is provided in Algorithm 2. At initialisation, the genetic information of the population, i.e. all reservoir weights matrices and parameters, are stored in memory. The population is then evaluated and given a fitness value which is stored. Fitness is defined here as the system's test set error; each individual therefore undergoes training, validation and testing during this process. At the validation stage, regularisation parameters are found to improve network generalisation to new data. In the test stage, the final trained network is then evaluated on new unseen data from the test set.

Algorithm 2 Microbial GA (M-GA)

```
1: Initialise population P
2: popFit = Evaluate(P)                                  ▷ popFit & P are stored
3: while !maxgens do
4:     Assign Indv1 and Indv2 randomly from P
5:     if popFit(Indv1) ≤ popFit(Indv2) then
6:         winner =Indv1, loser =Indv2
7:     else
8:         winner =Indv2, loser =Indv1
9:     P(loser) = Infect(P(loser), P(winner), recRate)
10:    P(loser) = Mutate(P(loser), mutRate)              ▷ Update P
11:    popFit(loser) = Evaluate(P(loser))                ▷ Update popFit
12:    print(min(popFit))
```

At every generation tournament selection occurs, where two individuals are randomly selected and their fitnesses are compared. The individual with the lower error is marked as the *winner* and the other the *loser*. The loser's genetic information is copied to create a new individual (the child) for further manipulation. At the infection stage one-way recombination is applied and controlled by *recRate*, replacing a proportion of the child's genetic information with information from the winner. The child's new genes are subject to mutation based on the mutation rate *mutRate*. Evaluation of the child is performed and the child then replaces the loser in the stored population. This loop repeats until the maximum number of generations are reached. The M-GA parameters used are given in Table 7.1. In total, 2000 fitness evaluations (population + generations) occur for each run and for every architecture to provide a fair comparison.

7.4.1 Mutation and Recombination

During the search process a variety of parameters are under manipulation. These fall into two categories: global sub-reservoir parameters, or local/inter-reservoir weights. The first consists of parameters for input scaling, internal matrix W_r scaling, and leak rate.

Parameter	Value
Generations	1985
Population size	15
Total Evaluations (per run)	2000
Runs	10
Mutation rate	0.3
Recomb./crossover rate	0.4
Deme size	14

Table 7.1: Hyperparameters for Microbial GA.

The second includes input weights, sub-reservoir weights and weights connecting sub-reservoirs. Two operators are defined to mutate each of these categories. For local sub-reservoir parameter mutation, the probability of mutating any sub-reservoir parameter is 1/3. If mutation occurs, new values for the leak rate are bound between [0 1], for the internal weight scaling [0 2], and for input scaling $[-1\ 1]$. For weight mutations (on the non-RoR architectures), there is a 0.25 probability for: adding a new internal weight (value between $[-0.5\ 0.5]$) at a random location, randomly adding an input weight (between $[-1\ 1]$), randomly removing an internal weight, and randomly removing an input weight. For the RoR architectures, these probabilities change as weight mutation can also occur in any connecting sub-reservoir matrix W_{ij} within the inter-reservoir matrix $\bar{W}$. The probabilities change to 0.5 of mutating any weights in each sub-reservoir, or within the inter-reservoir matrix.

As there are many evolvable parameters in each system, a limit on the number of mutations is imposed to reduce large destructive jumps in the search space. A maximum of m weight mutations can occur on any sub-reservoir, set as $m \approx 1\%$ of the total weight parameters. The mutation rate *mutRate* is therefore used to determine what percentage of m is mutated. The *mutRate* also determines whether local sub-reservoir parameter mutation occurs for each sub-reservoir at each generation.

The infection/recombination phase is similar across all architectures. However, when evolving a single network, infection completely replaces the loser with the winner. To reduce the frequency of this large change infection occurs only at generations determined by *recRate*. For example, if $recRate = 0.5$, infection will only occur at a generation 50% of the time. For the other architectures, infection can replace any sub-reservoir within the child with any from the winner. For the RoR architectures, this requires that the inter-reservoir matrix $\bar{W}$ also be updated.

In the full framework, additional categories and operators are also available: *neuron* mutation (add/remove neurons and associated weights) and *sub-reservoir* block mutation (add/replace/remove a sub-reservoir and its connections). However, these are ignored here as these mutation operators typically require additional fine tuning.

7.5 Benchmarks

The benchmark tasks used to assess the architectures are the Non-linear Auto-Regressive Moving Average (NARMA) tasks. The same task is outlined in Chapter 3.

The output $y(t)$ for the different n-th ordered systems applied here are constructed from (7.9), using input $u(t)$ generated from a uniform distribution of [0, 0.5]. Three n-th order systems are applied, each increasing in complexity. The parameters for the n-th ordered systems are: for the 10-th order $\alpha = 0.3, \beta = 0.05, \delta = 0.1$, and $\gamma = 1$; for the 20-th order $\alpha = 0.3, \beta = 0.05, \delta = 0.01$, and $\gamma = tanh()$; for the 30-th order system $\alpha = 0.2, \beta = 0.004, \delta = 0.001$, and $\gamma = 1$.

$$y(t+1) = \gamma(\alpha y(t) + \beta y(t)\left(\sum_{i=0}^{n-1} y(t-i)\right) + 1.5u(t-9)u(t) + \delta) \tag{7.9}$$

For every n-th ordered system a total of 8000 samples were used, split into 2000 training, 3000 validation and 3000 samples for testing, with a 100 sample washout period to remove the effects of initial zero states. Each dataset was subjected to mean normalisation and offset by -0.5. For each system, training was performed using ridge regression with Tikhonov regularisation with the regularisation parameter selected using validation data.

7.6 Results

7.6.1 Neuroevolution – A Single ESN

The experimental results in Table 7.2 demonstrate that the microbial GA can consistently find LI-ESNs that significantly outperform randomly generated ESNs. In fact, on average, evolved LI-ESNs produce errors below the lowest error of the best random reservoirs. To calculate averages, 10 runs were used with a maximum of 2000 fitness evaluations per run. For the LI-ESN (Rand.), this consisted of 2000 random initialisations per run, and for the LI-ESN (M-GA) 2000 fitness evaluations per run. The minimum errors shown are those from all runs.

As shown in Table 7.2, the LI-ESNs (M-GA) outperforms other state-of-the-art optimisation techniques in the literature. For example on the NARMA-10 task with 200-nodes, the LI-ESN (M-GA) significantly outperforms larger reservoirs such as the 300-node evolutionary pre-trained cycle reservoir with regular jumps (CRJ-E), and on average almost matches the 400-node Bayesian-optimised ESN (ESN-OSI).

The LI-ESN (M-GA) evolved at 400-nodes, significantly outperforms all of the other reservoir models and optimisation techniques demonstrating what appears to be the best reported performance for this task and network size.

7.6.2 Neuroevolution – Hierarchical ESNs

Results for the NARMA-10 and NARMA-30 tasks on all six evolved architectures under-test are shown in Figures 7.3a, 7.3b and Table 7.3. In this experiment, three network sizes (100-node, 200-node and 400-node) are tested to evaluate sub-reservoir size and number of sub-reservoirs in use. At 100-nodes, all multi-reservoir architectures are subdivided into two sub-reservoirs of 50-nodes each. At 200-nodes, each sub-reservoir increases to 100-nodes. At 400-nodes, four sub-reservoirs are created with 100-nodes each.

Although the number of neurons in each model is the same, a clear discrepancy in the number of possible connection weights arises. For example, for the 100-node LI-ESN (M-GA) a total of 10,000 (100×100) neuron weights exist, whereas for the 100-node

Reservoir Type	Units	NARMA-10		NARMA-20		NARMA-30	
		Mean (σ)	Min.	Mean (σ)	Min.	Mean (σ)	Min.
LI-ESN (Rand.)	200	0.3625(0.3128)	0.0105	0.5588(0.1981)	0.0920	0.6272(1.5507)	0.1529
LI-ESN (M-GA)	200	**0.0066**(8.9E-4)	0.0053	0.0677(0.0048)	0.0603	**0.1455**(0.0047)	0.1396
RCDESIGN [57]	174(16.72)	0.0084	-	-	-	-	-
ESN [158]	200	0.0425 (0.0166)	-	0.167 (0.0164)	-	-	-
DLRB [158]	200	0.0402 (0.0110)	-	0.160 (0.0153)	-	-	-
CRJ-E [206]	200	0.0200	-	-	-	-	-
CRJ-E [206]	300	0.0216	-	-	-	-	-
LI-ESN (M-GA)	400	**0.0026**(2.5E-4)	0.0021	-	-	**0.0940**(0.0125)	0.0761
ESN-OSI [204]	400	0.0064	-	-	-	-	-
GESN [156]	600	0.0051(5e-5)	-	-	-	-	-

Table 7.2: NARMA performance of LI-ESN(M-GA) compared to random LI-ESNs and other reservoir techniques. Standard deviation σ and minimum errors only provided when given in the literature.

RoR there is only 7,500 *neuron*-to-*neuron* weights ($3 \times (50 \times 50)$), 5000 for both sub-reservoirs and 2500 for the inter-reservoir connectivity. This means an RoR of this size features a 25% connection deficit in comparison to a fully connected, single network of “equivalent” neuron size. This continues to increase up to a 37.5% connection deficit for the 400-node RoR, and 75% deficit for the 400-node ensemble architecture.

The total number of *neuron*-to-*neuron* weights (excluding input and output weights) of an RoR system can be calculated as $n_r^2 \left(\frac{1}{2}n_s(n_s-1)+n_s\right)$, where n_s is the number of sub-reservoirs and n_r the number sub-reservoir nodes. Therefore, in order to roughly match the 200-node LI-ESN neuron-to-neuron weights using 2 layers would require 115-nodes in each sub-reservoir, i.e. an increase in nodes is needed in each sub-layer, meaning node compensation is always required when dividing a equivalent sized network into layers.

For both tasks, and all network sizes, the non-hierarchical LI-ESN (M-GA) appears to outperform all other architectures. This is not surprising given the connection deficit between the models. However on the NARMA-10 task, despite the deficit increasing with node size, the RoR-IA architecture and LI-ESN (M-GA) show no significant difference when using the non-parametric Wilcoxon rank-sum test. However, for the harder NARMA-30 task, the deficit and its effect on performance becomes apparent with the

Task	Units	Architecture					
		LI-ESN (M-GA)	RoR-IA	DeepESN-IA	DeepESN	RoR	Ensemble
NARMA-10	100	**0.0191**(0.0045)	0.0230(0.0046)	0.0268(0.0047)	0.0240(0.0044)	0.0334(0.0057)	0.0298(0.0032)
	200	**0.0066**(0.0009)	0.0071(0.0011)	0.0078(0.0017)	0.0086(0.0013)	0.0108(0.0021)	0.0086(0.0013)
	400	**0.0026**(0.0003)	0.0027(0.0002)	0.0032(0.0005)	0.0035(0.0006)	0.0046(0.0006)	0.0036(0.0005)
NARMA-30	100	0.1592(0.0180)	0.1604(0.0043)	**0.1581**(0.0032)	0.1609(0.0094)	0.1641(0.0102)	0.1697(0.0093)
	200	**0.1455**(0.0047)	0.1509(0.0037)	0.1509(0.0029)	0.1512(0.0027)	0.1517(0.0035)	0.1525(0.0023)
	400	**0.0940**(0.0125)	0.1178(0.0118)	0.1393(0.0135)	0.1489(0.0044)	0.1504(0.0034)	0.1507(0.0035)

Table 7.3: Results of evolved architectures for the NARMA-10 and NARMA-30 tasks. Mean NMSE and standard deviation are given for multiple architecture sizes, with the lowest highlighted. The LI-ESN outperforms others in most cases, with RoR-IA featuring the next lowest errors.

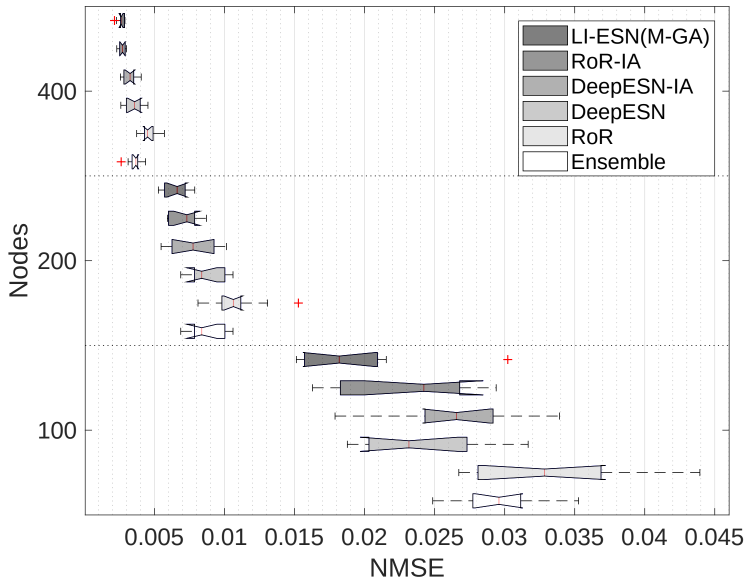
LI-ESN (M-GA) outperforming all others.

To demonstrate the weight deficit effect on performance, an additional experiment is conducted. The RoR-IA is evolved again using the 2-layer, 115-node *neuron*-to-*neuron* weight equivalent. The new weight matching network significantly outperforms the LI-ESN (M-GA) on the NARMA-10 task with a mean *NMSE* = 0.0053 and minimum *NMSE* = 0.0041. However, it is debatable whether this comparison is fair, as the weight matching RoR-IA has more neurons and thus more states to utilise.

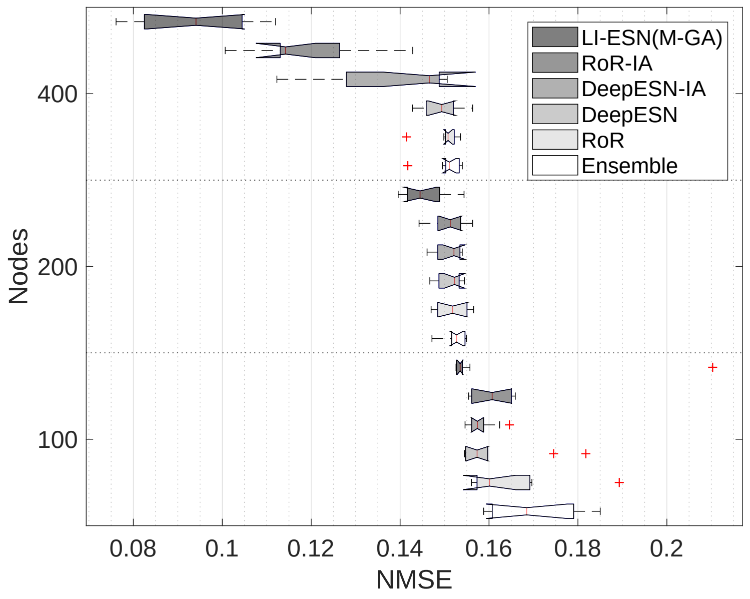
The results of all evolved architectures show clear improvements in performance compared to others in the literature. For example, on the NARMA-30 task all architectures with 100-nodes show a mean error lower than the 150-node evolved compositional pattern producing networks (CPPNs) reported in [133] (*NMSE* = 0.1696). And for the NARMA-10 task at 400-nodes, the means of all evolved architectures in Fig. 7.3a are lower than the evolved 600-node GESN in Table 7.2.

7.7 Task Generalisation

For any reservoir computer good performance often requires a trade-off between memory and non-linearity, as non-linearity inherently degrades memory. This trade-off is widely studied in reservoir computing [194]. A recent illustration of the dynamical mechanism behind it is shown in [85], as well as how adding "a pinch" of linear dynamics to non-



(a) NARMA-10 Task



(b) NARMA-30 Task

Figure 7.3: Task performance for each architecture. For all architectures (*except M-GA*), 100 nodes is equal to 2 sub-reservoirs of 50 nodes, 200 nodes $= 2 \times 100$ nodes, and 400 nodes $= 4 \times 100$ nodes. The single network (M-GA) produces the lowest errors, followed closely by RoR-IA.

linear reservoirs can significantly improve reservoir systems.

Hierarchical structures offer a way to decouple non-linearity and memory, removing the trade-off problem. A decoupled structure has the potential to exhibit multi-scale dynamics that could be utilised for multiple tasks. Hierarchical networks therefore, in theory, have the potential to have greater network generalisation to other tasks, as generalisable features may emerge.

To test the task generalisation hypothesis, two additional benchmark tasks are evaluated: the Santa Fe laser time-series prediction task and the Hénon map task. The experiment is therefore to evolve an architecture to some task then evaluate its ability to perform a new task it was not evolved for. If the hierarchical/modular reservoirs consistently generalise better to the new task than does the single network, we can infer some potential benefit from structural differences.

The Santa Fe laser dataset is the same applied in Chapter 4. The task is to predict the next value $u(t+1)$ based on the current input $u(t)$. Training and testing is split into three sets: 2500 samples for reservoir weight training, 1250 validation, and 1250 samples for testing with a washout of 100 samples.

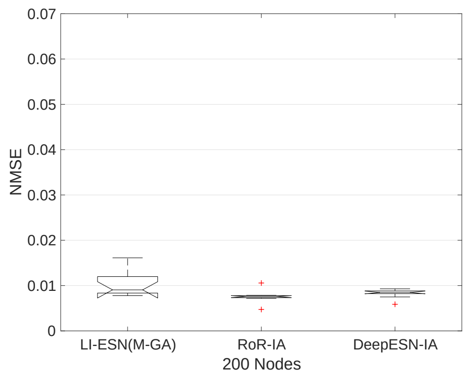
The *Hénon map* [77] task is featured in both reservoir computing and recurrent neural networks. The task is to predict the next value $y(t+1)$ of a two-dimensional mapping of a strange attractor, given by:

$$y(t) = 1 - 1.4y(t-1)^2 + 0.3y(t-2) \tag{7.10}$$

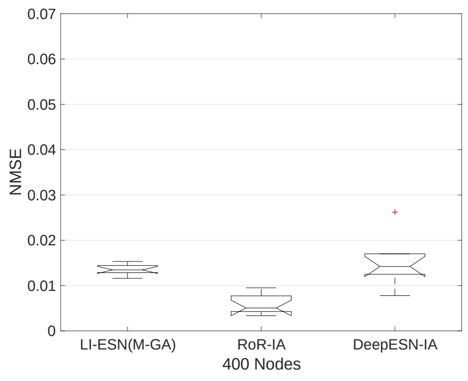
To increase the difficulty of the task, Gaussian white noise with a standard deviation of 0.05 is added to $y(t)$. The dataset for this task is split into: 2000 training, 3000 validation and 3000 testing, with a washout period of 100 samples for each.

In the following experiment, the two best performing hierarchical networks from the previous experiment are used as well as the non-hierarchical LI-ESN (M-GA) network. For each architecture, the best evolved network from each run is re-trained to each new task; re-training applies only to the output weights, no other parameters are changed.

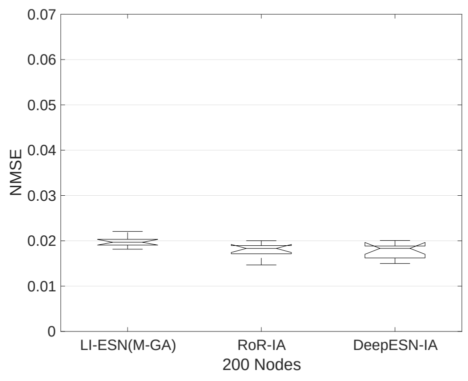
The results show the RoR-IA architecture generalises better to every task compared to the LI-ESN (M-GA), despite size and original task. Fig. 7.4 shows each instance of the experiment split into network size and original evolved task. This suggests the RoR



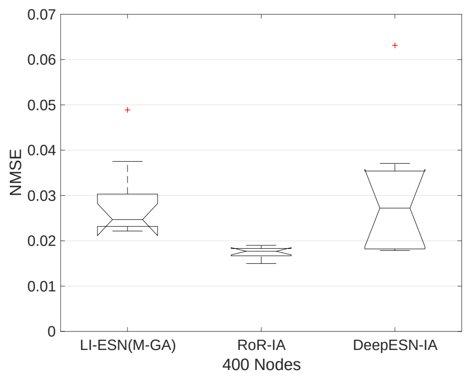
(a) Laser Task- 200 node (NARMA-10)



(b) Laser Task- 400 node (NARMA-30)



(c) Hénon Map - 200 node (NARMA-10)



(d) Hénon Map - 400 node (NARMA-30)

Figure 7.4: Results of evolved networks retrained to laser and Hénon map tasks.

structure helps improve generalisation, which also appears to increase with network size. RoR is also the only architecture to perform better when using more nodes, whereas for the other two performance typically decreases.

The DeepESN-IA, however, shows no significant difference within the 95% confidence level from the non-hierarchical LI-ESN (M-GA). This might suggest that hierarchy alone does not necessarily always lead to improved generalisation.

Understanding which hierarchies and what dynamical properties of each sub-network improves generalisation still requires further investigation. This is an interesting avenue for future work as the task generalisation results in Fig. 7.4 show retrained networks can outperform other reservoir models, and even match some reservoirs optimised directly to the test task. Reported averages using 200 nodes in the literature are: $NMSE = 0.0082$ for an ESN in [158] and an $NMSE = 0.0066$ for evolutionary pre-trained CRJ in [206]. The

RoR-IA with 200 nodes, despite being trained on another task, has an average *NMSE* = 0.0075 and the best network gives *NMSE* = 0.0047.

7.8 Future Work with RoRs

In [124, 125], separate projection and encoding layers are defined to maximise feature abstraction and break the collinearity problem. Each encoding layer produces a low-dimensional representation that projects into the next layer. The evolved solutions in this chapter potentially do not take advantage of cascading high-dimensional projections, as representations generated in the first sub-reservoir can project into a state space of the same dimension. However, in these experiments, connectivity between sub-reservoirs is typically low and sparse (see Fig. 7.5), potentially allowing multi-scale projections between sub-reservoirs. To better understand this requires further investigation.

An interesting experiment for future work would be to combine different reservoir architectures. Sub-reservoirs could be constructed from different neural network architectures and reservoir computing systems. The reason why one might want to do this is to exploit unique dynamical characteristics of different architectures. The potential for combining different architectures can be seen in the literature, where examples include adding and combining feed-forward layers and Extreme Learning Machines (ELM) [26, 27], and adding encoding layers [124, 125].

Other avenues for future work include: i) a thorough analysis of the evolved weights to assess the homogeneity of the network. As explained in [92] with HESNs, more connections between sub-reservoirs often leads to more homogeneous networks. As briefly shown in Fig. 7.5, which tends to be a typical case for RoRs, the networks are more heterogeneous than homogeneous. ii) further tuning of the microbial GA could lead to increases in performance.

7.9 Summary

In this chapter, the Reservoir-of-Reservoirs (RoR) architecture is described; it comprises a master reservoir containing smaller evolved sub-reservoirs. The first results

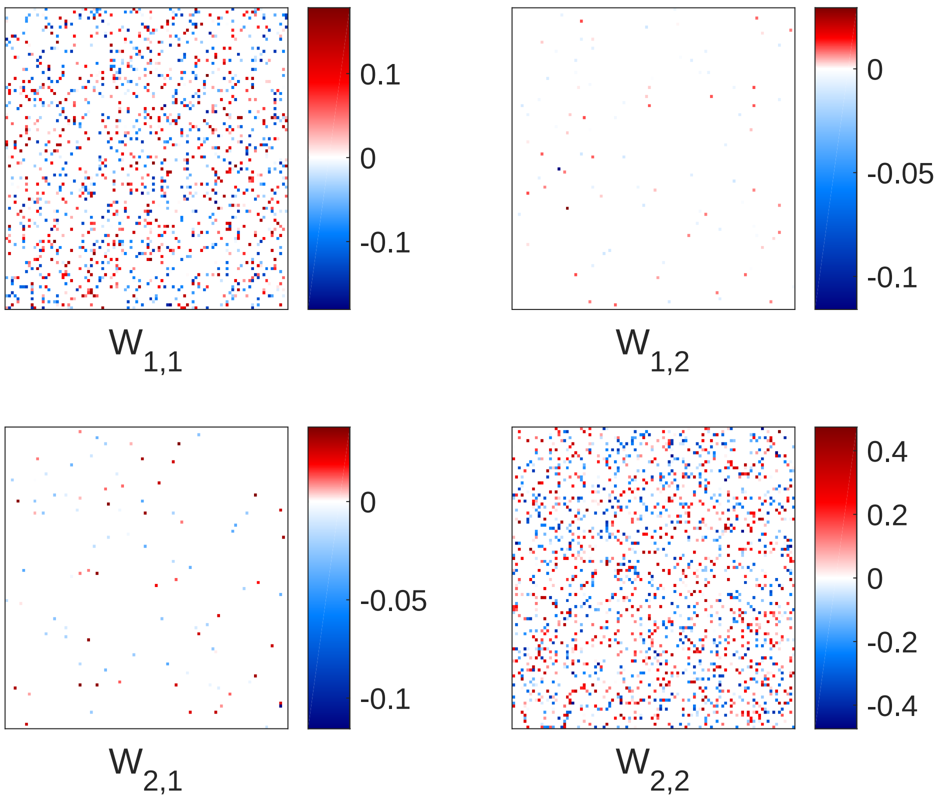


Figure 7.5: Weight matrices plots of an evolved 200 node RoR-IA network. Matrices $W_{1,1}$ and $W_{2,2}$ represent the internal connectivity/weights of the two 100 node sub-reservoirs. Matrices $W_{1,2}$ and $W_{2,1}$ show connectivity between the two sub-reservoirs.

showed the microbial GA outperforms not only random search but significantly outperforms other ESN optimisation techniques using smaller networks. This result is encouraging, as the microbial GA's implementation is very simple, making it easy to transfer to other systems.

The main results showed the RoR works best when the input is presented to every sub-reservoir. However, all of the evolved hierarchies typically fall short of the performances found with evolved non-hierarchical ESNs. This is hypothesised to be a consequence of the difference in the number of connections between models, with some models having up to 75% fewer weight connections. It is shown that if connectivity is compensated for, by adding additional nodes, the evolved hierarchies perform very well in comparison, and improve upon much larger reservoir systems in the literature. This result could help inform design decisions and considerations when moving to the physical domain. For example, forming a large reservoir from many smaller reservoirs will undoubtedly feature the same connectivity problem, but this might be negated by the impracticality of creating such large, equivalent sized physical reservoirs anyway.

In the final experiment, the RoR's ability to learn generalised features is tested by retraining evolved networks to new tasks. The results show the RoR provides significant improvements over other tested architectures, suggesting the RoR consistently evolves some generalised features that others do not. This result raises some interesting questions and possibilities. If consistent, it could help prove the potential of a more general computing multi-substrate reservoir computer.

Although encouraging results are shown, a current drawback of this chapter is that too few experiments are conducted to observe what effect the number of sub-reservoirs has on performance. In fact, little is still known about what the ideal ratio of sub-reservoir neurons to number of sub-reservoirs is. Furthermore, it is still unknown whether more, or fewer, sub-reservoirs are advantageous or detrimental. Further experiments and analyses are still needed to answer these fundamental questions.

In the next chapter we take a step back and try address a basic, but non-trivial problem; how to determine whether a substrate is suitable for reservoir computing. To fully exploit the mutli-reservoir architecture proposed in this chapter many new substrates may be necessary, featuring different but complementary properties. How to find and compare such reservoirs is still unknown. For this reason, in the next chapter, we propose a generic framework to assess and compare the quality of any substrate for reservoir computing.

Chapter 8

Characterising Substrate *Quality* for Reservoir Computing

In recent years, the reservoir computing framework has been applied to a variety of physical systems such as, optoelectronic and photonic [10, 191], quantum [59, 151, 189], disordered and self-organising [40, 184], magnetic [155], and memristor-based [51] computing systems. The way each substrate realises a reservoir computer varies. However, each tends to implement, physically or virtually, a static network of coupled processing units.

Each implementation is designed to utilise and exploit the underlying physics of the substrate, to embrace its intrinsic properties to improve performance, efficiency and/or computational power. As with many physical systems, each can be configured, controlled and tuned to perform a desired functionality. In all of the above examples, this requires the careful tuning of parameters in order to produce working and optimal physical reservoirs.

In general, the abstract term *reservoir* usually represents a single, typically static, configuration of the substrate. For an artificial recurrent neural network, implemented *in silico*, this may refer to a set of trained connection weights, defined neuron types and topology. For another substrate, configuration may refer to the physical morphology, physical state, external control signals, or complexification of the driving input signal. This implies that the number of possible reservoirs realised by one substrate depends upon the number of free parameters and distinct dynamical behaviours resulting from those parameters. For unconstrained substrates, limited only by the laws of physics, this number may be vast. Yet, this does not imply that every configuration/reservoir is practical or useful.

In terms of all possible reservoirs realisable by one substrate, the vast majority may be unusable in terms of solving a task. However, some region of the substrate's configuration space may well provide interesting reservoirs and potentially high-performing reservoirs, or even reservoirs with large generalising computing abilities.

Characterising the configuration and reservoir space (referred to below as the *behaviour* space) of usable and optimal reservoirs would thus help describe the substrate's "quality" for reservoir computing, that is, the substrate's ability to realise different reservoirs, and therefore its capacity as a generic reservoir computing substrate.

According to [43], all dynamical systems have an almost universal characteristic to perform useful information processing, provided a fading memory and linearly independent internal variables are present. However, each dynamical system tends to suit different tasks, and rarely, but not unattainably, will one feature a universal set of properties to perform well across many, if not all tasks. This implies that high-performing, task-specific, and potentially some good task-generalising reservoir computers can be built. Then, when combined with highly-efficient unconventional substrates, these unique and powerful new computers can become a disruptive force.

8.1 How Do We Measure Quality?

Dambre [43] devises a quantitative measure which is independent of physical realisation, allowing anyone to compare the computational properties of a broad class of dynamical systems. However, that total capacity measure may not be informative enough to guide substrate optimisation for specific tasks, or physically demonstrate how expressive the substrate is in terms of realising vastly different reservoirs.

So far, no practical framework exists to map, then utilise, the full computational expressiveness of physical or virtual substrates. That is, no experimental method has been proposed to characterise the reservoir computing *quality* of substrates, or to use measures of computational properties to configure and discover optimal reservoirs.

The challenge in creating a generic framework arises from the enormous variety of

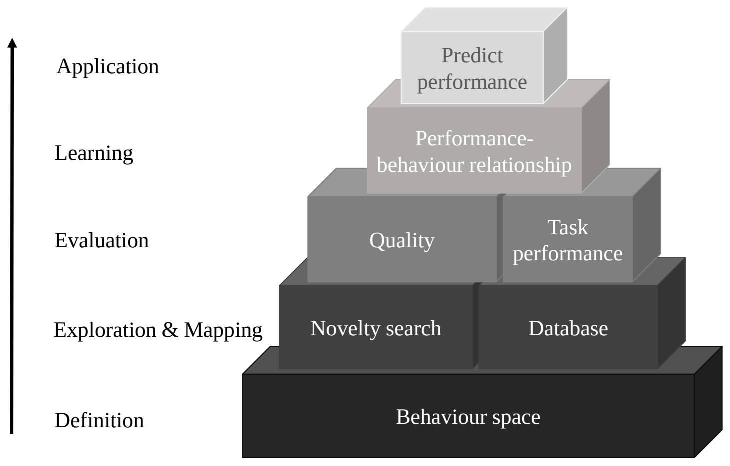


Figure 8.1: Framework levels and building blocks.

possible substrates, and from each substrate having its own set of configuration and other parameters. The term "substrate" is used here to refer to any physical or virtual system that realises a reservoir computer, e.g. a physical electronic or optical circuit, a virtual gene regulatory network, or a cellular automaton; essentially anything featuring configurable parameters and a method to observe system states.

To tackle this non-trivial problem the SQuARC (Substrate Quality Assessment for Reservoir Computing) framework is proposed. The main purpose of the framework is to characterise and assess the quality of any potential RC substrate. This attempts to complement previous theoretical work in the RC community, such as [43]. The framework also has a secondary purpose, to utilise the quality assessment process to better understand the general relationships between computational properties and task performance.

8.2 The SQuARC Framework

To conceptualise and visualise the framework it is divided into a series of building blocks and levels, as shown in Fig. 8.1. To make use of the framework, and to assess accuracy and validity, five stages are to be completed. These are represented by the five levels: 1) *Definition*, 2) *Exploration & mapping*, 3) *Evaluation*, 4) *Learning*, and 5) *Application*.

Each level of the framework features a reliance on the level below: to start a new

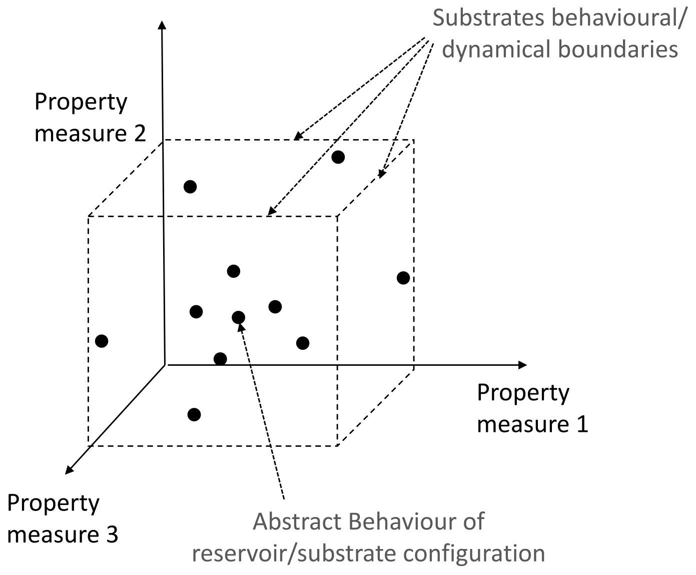

Substrates behavioural/ dynamical boundaries
Property measure 2
Property measure 1
Property measure 3
Abstract Behaviour of reservoir/substrate configuration

Figure 8.2: Example of a 3-Dimensional Behaviour Space. Here each abstract behaviour is relative to the three chosen property measures. Given enough time to explore the space, the substrate's dynamical/behavioural limitations become apparent.

level requires the completion of the level below. Each level consists of building blocks. In general, these blocks are adaptable and may be improved. For example, the current method for exploring the behaviour space may be improved or interchanged with another exploration technique. Another example is that more dimensions can be added to the behaviour space with the creation of new property measures. In some cases, as will be explained later, some blocks/levels can even be removed.

8.2.1 Definition

On the first level, a *behaviour* space is defined. This abstract behaviour space represents the dynamical behaviour of the substrate when configured. To represent the *n*-dimensional space, *n* independent property measures are used, defining the axes of the space (see example in Fig. 8.2). It is hypothesised here that the more distinct measures applied, the better the representation of the substrate's dynamical freedom and thus the better the accuracy of the quality measure.

8.2.2 Exploration & Mapping

The next level is *Exploration* & *Mapping*. To determine a *true* measure of quality, not only an approximation, exploration of the behaviour space would require an exhaustive search of the substrate's parameter space, which is infeasible. Rather than exhaustive search, an implementation of novelty search (NS) [107] is recommended. Novelty search is an open-ended genetic algorithm designed to explore a behaviour space for novel solutions until some user-defined termination criteria.

Rewarding novelty can maximise exploration by promoting diversity into new areas of the search space. A key advantage over objective-based search is that novelty search can be particularly useful when the relationship between parameters and behaviours is deceptive [107], for example, when a strong selection pressure can in fact inhibit innovation.

When applied in this manner, exploration characterises the substrate's search space and as a by-product outlines its dynamical boundaries, when given enough time. This experimental characterisation can then help determine the practical use, if any, of the substrate, or whether the selected method of configuration and observation (which itself may be optimised) is appropriate.

To make later use of the exploration process every behaviour found throughout the search is stored in a *database*. The NS algorithm itself is designed to perform a global search of the space: exploration rather than exploitation. However, across the generations many smaller local searches are also occurring in the process. The database therefore keeps a record of all behaviours, including localised behaviours around a novel behaviour.

8.2.3 Evaluation

The next level is *Evaluation*. To determine quality, the spread and number of distinct behaviours recorded in the abstract behaviour space (the database) is quantitatively measured. Quality is therefore a measure of how many distinct dynamical behaviours the substrate experimentally possesses. The measure itself can be simple, however, any measure of quality requires some contextual reference. To assess the quality of a new substrate it is recommended that an easy to define/inspect, ideally known to be high-performing, reference substrate first be assessed and evaluated. In the work here, to provide a baseline substrate to compare to, and to evaluate the framework, simulated ESNs are used as a *reference* substrate. In the future, if better, i.e. good generalising substrates with higher degrees of dynamical freedom, are found, the reference substrate can be replaced.

The second building block on the evaluation level represents the task evaluation process when the database is assessed on specific tasks. Although not necessary to evaluate the quality of a substrate, this process is required to utilise the top two levels of the framework.

8.2.4 Learning and Application

The next two levels (*Learning* and *Application*) fulfil the secondary goal of the framework to relate computational properties and behaviours to performance. These two levels also provide a method to validate the substrate-independence of the framework. As a bonus, they also provide a method to significantly reduce the cost of future task assessments by predicting task performances based on similarly behaved substrates.

To validate the framework, a conjecture from Abstraction/Representation theory [81] is followed hypothesising that a faithful abstract representation should provide an abstract prediction of how the system will evolve. For example, in this scenario, if the behavioural representation is faithful across substrates and the relationship between properties and performance can be faithfully abstracted, it may be possible to predict performance of one substrate based on another. However, to predict performance across systems requires us to solve another challenging problem.

As explained in [65], relating properties to expected performance is non-trivial for all task applications, as good properties for one task may be detrimental to another. Therefore, no single set of properties will always lead to high performance. However, the relationship between properties and a single task may be simpler to determine.

To model and estimate the statistical relationships between properties and task performance a broad range of properties and performances are required. Novelty search provides property diversity in the exploration and mapping level, discovering reservoirs with (hopefully) very different dynamical properties. At the evaluation level task performances can be assessed; this leads to a basic dataset to train a learning system, for example, a neural network.

Building and training the learning system represents the *learning* level of the framework. This is again adaptable, as different models can be used to learn the relationships. At this level, the accuracy of the prediction implies how well the relationship between properties and performance can be abstracted. The resulting prediction accuracy also helps approximate the accuracy of the property measures and faithfulness of the behavioural representation. For example, if no relationships between properties and performance were found there would be little confidence in the ability of the property measures to represent the true computational properties of the system.

At the final *Application* level, the full validation of the framework is achieved. This is attained by measuring the difference in prediction accuracy between the reference and new substrate. A small difference would validate the accuracy of the learned models, the behaviour representation and substrate-independence of the framework. Beyond validation, this level is also useful to predict the task performance of any substrate – without the need to evaluate directly – based on the reference substrates behaviours and task performances.

8.3 Task-Independent Properties

In order to create the behaviour space, each axis/dimension of the space must be defined. In the following sub-sections different computational properties and measures are discussed.

A potential problem in defining the behaviour space is that some properties are difficult, if not impossible, to measure across all substrates. It is therefore important to remember that any properties applied with the framework should represent the behaviour of the system independent of its implementation.

The basic framework demonstrated later in the chapter applies only a subset of the properties discussed here. This section is intended to inform the reader of the various properties available in the RC community and potentially inspire the creation, or adoption from other fields, of new property measures for this framework.

8.3.1 Chaos and Criticality

Chaotic behaviour in dynamical systems can be characterised by sensitivity to perturbation and initial conditions. This is popularly measured by estimating the rate of divergence between close trajectories using the spectrum of Lyapunov exponents. Typically, the maximal Lyapunov exponent is preferred in reservoir computing. However, estimating the Lyapunov exponents can be non-trivial for high-dimensional noisy systems [120] and impractical in many cases for physical systems.

Chrol-Cannon & Yin [33] show that the maximal Lyapunov exponent and kernel quality (a measure of the *separation* property, described later) often strongly correlate, arguing that both capture similar dynamical properties of the reservoir. This makes observing a critical phase easier when Lyapunov exponents are difficult to estimate, or cannot be estimated. However, this relationship is found to break down when the number of distinct reservoir states required to separate the input classes has been reached.

The general consensus in reservoir computing is that Lyapunov exponents are good predictors for optimal performance across many tasks investigated [16, 25, 195, 196]. However, often a critical state by itself is only a weak indicator of performance across many task domains. For some tasks, high performance does coincide with a critical balance between chaotic and ordered behaviour, described by Verstraeten & Schrauwen [195] as a region between the excitability of the reservoir and its degrees of freedom in the state space. Being in a critical state, however, close to the *edge of chaos* is not a necessity. Some tasks require very different levels of computational dynamics, or even minimum levels of computation. Therefore, maximising any computational property can often be misleading and unnecessary.

Despite its limitations, the edge of chaos theory can, and has, played a useful role in quantifying reservoir performance and optimisation in simulated networks. The same concepts could also have direct connotations to material computation whereby a material can exhibit “richness”, and therefore be exploitable, only by operating close to or within this phase transition. A novel example of this can be seen where a self-organising structure of carbon nanotubes evolves to produce maximum entropy given a strong applied electric field [14]. The same proposition has only recently been considered in the field of evolution *in materio* with physical substrates [146], with more substantial investigations still needed.

An encouraging thought is that emergent criticality and self-organising properties might be commonplace in dynamic networks [181], suggesting many potential reservoir computing systems.

8.3.2 Kernel Quality and Generalisation Rank

Kernel quality is a measure of the reservoir's ability to produce a rich non-linear representation of input u and its history $u(t-1), u(t-2), \ldots$. Also known as the *linear separation property*, it was first introduced by Legenstein & Maass [105] to measure a reservoir's ability to separate distinct input patterns. As many practical tasks in machine learning are linearly inseparable, reservoirs would not be able to solve such problems without some non-linear transformation of the input.

The kernel quality measure is performed by computing the rank r of an $n \times m$ matrix M, outlined in [25]. To create the matrix M, apply m distinct input streams $u_i, ..., u_m$ and collect the resulting reservoir states x_{u_i}. Place the states x_{u_i} in each column of the matrix M and repeat m times. The rank r of M is computed using Singular Value Decomposition (SVD) and equal to the number of non-zero diagonal entries in the unitary matrix. The maximum value of r is always equal to the smallest dimension of M. To calculate the effective rank, and better capture the information content, remove small singular values using some high threshold value. To produce an accurate measure of kernel quality m should be sufficiently large, as accuracy will tend to increase with m until it converges.

The generalisation rank is a measure of the reservoir's capability to generalise given similar input streams. It is calculated using the same rank measure as kernel quality, however each input stream $u_{i+1}, ..., u_m$ is a noisy version of the original u_i. A low generalisation rank symbolises a robust ability to map similar inputs to similar reservoir states.

Reservoirs in ordered regimes typically produce low ranking values in both measures, and both are high when in chaotic regimes. In general, it is said that a good reservoir should possess a high kernel quality rank and a low generalisation rank [25]. However, in terms of matching reservoir dynamics to tasks, the right balance will vary. These two measures are important but by themselves do not capture enough information about the reservoir's dynamical properties.

8.3.3 Memory Capacity

A simple measure for the linear short-term memory capacity (MC) of a reservoir was first outlined in [87] to quantify the *echo state* property. For the echo state property to hold, the dynamics of the input driven reservoir must asymptotically wash out any information resulting from initial conditions. This property therefore implies a fading memory exists, characterised by the short-term memory capacity.

To evaluate memory capacity of a reservoir, how many delayed versions of the input $u(n-k)$ the outputs can recall, or recover with precision is measured. Using Eqn. (8.1), memory capacity is measured by how much variance of the delayed input is recovered, summed over all delays. This is carried out by training individual output units $O = \{O_1, O_2, O_3, \ldots, O_{N\times 2}\}$ to recall the input u at time k, i.e. $O_k = u(n-k)$.

$$MC = \sum_{k=1}^{O} MC_k = \sum_{k=1}^{O} \frac{cov^2(u(n-k), y(n))}{\sigma^2(u(n))\sigma^2(y(n))} \tag{8.1}$$

Jaeger [87] demonstrates that echo state networks driven by an i.i.d. signal can possess only $MC \leq N$, where N is the number of nodes.

A full understanding of a reservoir's memory capacity cannot be encapsulated through a linear measure alone, as a reservoir will possess some non-linear capacity. Other memory capacity measures proposed in the literature quantify the non-linear, quadratic and cross-memory capacities of reservoirs [43].

8.3.4 Class separation

Class separation is a metric that corresponds directly to different classes of input stimuli. Demonstrations of class separation can be found in [33, 63, 150]. Separation is measured as the average distance between resulting states, once again, given the assumption that significantly different inputs should generate significantly different reservoir states. To calculate separation requires the division of the input and state vectors into discrete classes; [63] provides an alternative measure characterised on the original assumption. For example, given two different input vectors $u_j(n)$ and $u_k(n)$ the euclidean distance between inputs should be large and positive, as described by D:

$$D := \|u_j(n) - u_k(n)\| \tag{8.2}$$

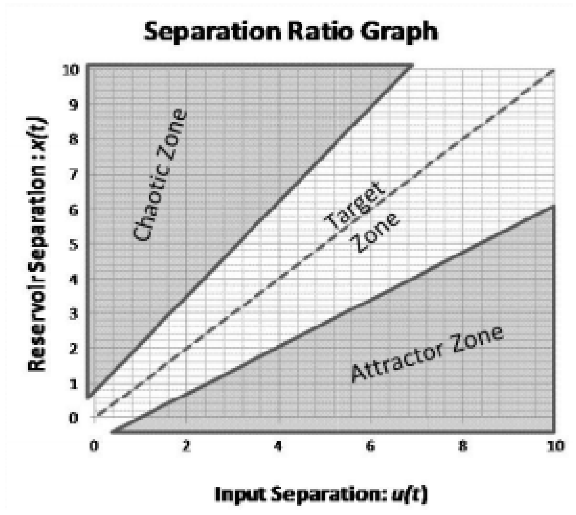


Figure 8.3: Separation Ratio Graph [63]. Graphical representation of the phase transition between chaos and order. Systems in the target zone possess both a good separation property and ideal dynamic behaviour to produce optimal reservoirs.

If the reservoir exhibits a good separation property the reservoir states $x_j(n)$ and $x_k(n)$ should increase in distance, or be equal to the original distance:

$$D \leq \|x_j(n) - x_k(n)\| \tag{8.3}$$

which can be represented as the ratio:

$$\frac{\|x_j(n) - x_k(n)\|}{\|u_j(n) - u_k(n)\|} \geq 1 \tag{8.4}$$

This simplified measure has been extended into *Separation Ratio Graphs* to produce a visual representation of separation and the phase transition of correlated dynamic behaviour (see Fig. 8.3).

Konkoli & Wendin [99] offer another comparable method for identifying reservoir quality in memristor networks. This metric is again based on the assumption that quality can be measured by observing the reservoir's ability to generate different dynamic states at the output. In this case, it is observed by measuring the dissimilarity between output states and a linear combination of the inputs, i.e. determining if the non-linear frequency response of a network cannot be approximated by a linear mixture of delayed inputs. Dissimilarity is measured in the Fourier space (ω) between outputs $o(n)$ and a linear combination of the time shifted inputs $z(n)$, given by:

$$\delta = \frac{\| o(\omega) - z(\omega) \|}{\| o(\omega) \|} \tag{8.5}$$

A large dissimilarity (large δ) is ideal in a reservoir as it describes a complex projection of the input. A small δ on the other hand may simply describe a linear propagation of the input, highlighting the absence of richness in the reservoir.

8.3.5 Information Theory

Another way to measure computational properties of dynamical systems is to utilise information theory [37, 169]. Information theory can describe the transmission, processing and storage of information. As a broad simplification, computation can be described as operations performed on information, or a manipulation of information. How much information is transferred between components during operations, the bandwidth of the communication and the internal storage of each component can help us quantify the dynamical properties of the system.

Shannon entropy, a key measure in information theory, estimates the average uncertainty of any measurement x of a random variable X. The typical measurement of entropy is in units of bits, using the base two logarithm as defined in:

$$H_X = -\sum_x p(x) \log_2 p(x) \tag{8.6}$$

Other important measures are as follows: The joint entropy of two random variables X and Y is a generalization to quantify the uncertainty of their joint distribution:

$$H_{X,Y} = -\sum_{x,y} p(x,y) \log_2 p(x,y) \tag{8.7}$$

The conditional entropy of X given Y is the average uncertainty that remains about, or the amount of information needed to describe the outcome, x when y is known:

$$H_{X|Y} = -\sum_{x,y} p(x,y) \log_2 p(x|y) \tag{8.8}$$

The mutual information between X and Y measures the average reduction in uncertainty about x that results from learning the value of y, or vice versa:

$$I_{X;Y} = H_X - H_{X|Y} = H_Y - H_{Y|X} \tag{8.9}$$

The conditional mutual information between X and Y given Z is the mutual information between X and Y when Z is known:

$$I_{X;Y|Z} = H_{X|Z} - H_{X|Y,Z} \tag{8.10}$$

In [16, 113, 114, 165], these measures are used to quantify properties of complex systems such as Local Active Information Storage (AIS) and Transfer Entropy (TE).

Information-theoretic measures can be used to describe the process by which each reservoir node X updates or computes its next state, with each computation utilising information storage from the node itself, and information transfer from other nodes.

The information storage of a node is the amount of past information present relevant to predicting its future state. Excess entropy, as formalised by Lizier *et. al.* [114], is a measure of the total information storage used in the future of a process. Lizier *et. al.* also describe how information storage in a node's environment within a distributed computation increases its information storage capacity beyond its internal capability. This effectively explains how nodes can utilise stigmergy and store extra information in neighbours and the rest of the network to be retrieved later. Some information in this process, however, will not necessarily be used at the next time step. Therefore, knowing how much stored information is used to compute the next value at the next time step is useful. This is described as the *active* information storage A_X.

This AIS for a node X is defined as the average mutual information between its semi-infinite past $x_n^{(k)} = \{x_n, x_{n-1}, \ldots, x_{n-k+1}\}$ and its next state x_{n+1}:

$$A_X = \lim_{k\to\infty} \sum_{x_{n+1}, x^{(k)}} p(x_{n+1}, x^{(k)}) \log_2 \frac{p(x_n^{(k)}, x_{n+1})}{p(x_n^{(k)})p(x_{n+1})} \tag{8.11}$$

Another useful measure is Transfer Entropy (TE) [113, 165] representing the dynamic information transfer between a source and a destination node. This is defined as the information provided by the source about the destination's next state that was not contained in the destination's past.

Transfer entropy is measured as the mutual information between the previous state of the source node Y to a destination node X. This is formulated as the source node y_n and the next state of the destination x_{n+1}, conditioned on the semi-infinite past of the destination $x_n^{(k)}$ (as $k \to \infty$):

$$T_{Y\to X} = \lim_{k\to\infty} \sum_{\mathbf{u}_n} p(u_n) \log_2 \frac{p(x_{n+1} \| x_n^{(k)}, y_n)}{p(x_{n+1} \| x_n^{(k)})}, \tag{8.12}$$

where u_n is the state transition tuple $(x_{n+1}, x^{(k)}, y_n)$ and $T_{Y\to X^{(k)}}$ represents finite-k approximation.

To make quick use of these measures, toolboxes such as the *Java Information Dynamics Toolkit* (JIDT) [112] are recommended.

8.4 Behaviour Exploration

In general, theoretically determining the computational *capacity* of a system helps us understand its limitations. One might think that this knowledge should then be used to construct or search for reservoirs with a maximal computational capacity. However in practice such maximisation is often unnecessary, time-consuming and may in fact hinder performance. A balance between properties is essential to match reservoir dynamics to tasks.

The SQuARC framework starts by exploring and mapping a vast range of dynamics for which the right balance can be selected for any task. In order for the framework to function properly and translate to many systems, this mapped space of dynamics requires substrate-independence.

To be substrate-independent, exploration must function without any prior knowledge of how to construct reservoirs far apart from each other in the behaviour space. Exploration cannot, therefore, be measured in the substrate parameter space; diversity in dynamics does not always coincide with diversity in parameters.

8.4.1 Objective-Based Evolutionary Algorithms

Exploration with objective-based search methods has a popular history, however at times it can be tricky and difficult to implement universally. In general, objective-based search focusses on an optimisation problem involving the minimisation or maximisation of single or multiple objectives. The exploration process depends heavily on the defined objectives and constraints.

In many approaches with evolutionary algorithms, exploration is characterised by promoting population diversity, the distance between genotypes or phenotypes, or fostering niches through selection [23, 64], for example, optimising the ratio of the number of unique individuals over population size, or the Hamming distance between genotypes represented by binary strings. Improving population diversity, in general, is often viewed as a way to directly tune the exploration/exploitation trade-off, biasing the evolutionary process in favour of more exploration.

In some multi-objective evolutionary algorithms, diversity of the population becomes an objective itself [45, 188]. These Pareto-based algorithms, often searching for a non-dominated set of solutions spanning the objective space (called the Pareto front [46]) can be programmed to obtain not only Pareto-optimal trade-offs between the original objectives, but also diverse solutions spanning the Pareto front.

A disadvantage of using this technique for the SQuARC framework is the loss of potentially interesting solutions that do not lie on the Pareto front. In many ways, an objective-based search does not necessarily align with the type of exploration wanted, in particular, where total coverage of the space is desired, not just optimal solutions.

8.4.2 Novelty Search

For these reasons, an open-ended evolutionary algorithm called novelty search (NS) [107, 108, 110] is adopted. In this particular implementation, novelty search is used to characterise the substrate's search space, i.e. the dynamical freedom of the substrate, by sampling its most interesting dynamical behaviours.

In contrast to objective-based techniques, a search guided by novelty has no explicit task-objective other than to maximise novelty. Novelty search directly rewards divergence from prior behaviours instead of rewarding progress to some objective goal.

Exploration without objectives has been shown, somewhat counter-intuitively, to outperform objective-based methods with deceptive task and solution spaces [157]. A deceptive objective (fitness) landscape is one where local optima are pervasive. When characterising substrate dynamics, this is of particular concern due to the high dimensionality of the substrate's computational properties which are only partially described, i.e. measures are only approximate.

Novelty search explores the behaviour space by promoting configurations that exhibit novel behaviours. Novelty of any individual is computed with respect to its distance from others in the behaviour space. To track novel solutions, an *archive* is created holding previously explored behaviours. Contrary to objective-based searches, novelty takes into account the set of all behaviours previously encountered, not only the current population. This enables the search to keep track of (and map) lineages and niches that have been previously explored.

To promote further exploration, the archive is dynamically updated with respect to two parameters; ρ_{min} and an update interval. The ρ_{min} parameter defines a minimum threshold of novelty that has to be exceeded to enter the archive. The update interval is the frequency at which ρ_{min} is updated. Initially, ρ_{min} should be low, and raised or lowered if too many or too few individuals are added to the archive in an update interval. Typically in other implementations, a small random chance of any individual being added to archive is also set.

In the following implementation, a small initial ρ_{min} is selected relative to the behaviour space being explored and updated after a few hundred generations. ρ_{min} is dynamically raised by 20% if more than 10 individuals are added and ρ_{min} is lowered by 5% if no new individuals are added; these values are guided by the literature.

To maximise novelty, a selection pressure rewards individuals occupying sparsely populated regions in the behaviour space. To measure local sparsity, the average distance between an individual and its k-nearest neighbours is used. A region that is densely populated results in a small value of the average distance, and in a sparse region, a larger value. The sparseness ρ at point x is given by:

$$\rho(x) = \frac{1}{k}\sum_{i=1}^{k} dist(x, \xi_i) \tag{8.13}$$

where ξ_i are the k nearest neighbours of x.

The search processes is guided by the archive contents and the current behaviours in the population, but the archive does not provide a complete picture of all the behaviours explored. Throughout the search process the population tends to meander around existing behaviours until a new novel solution exceeding the novelty threshold is discovered. To take advantage of this local search, here all the explored behaviours are stored in a separate database *D*. The database therefore stores all the information to later characterise the substrate and has no influence on the search, which uses only the archive.

8.4.3 Novelty Search Implementation

In the literature, novelty search is frequently combined with the Neural Evolution of Augmented Topologies (NEAT) [110, 174] representation; this neuro-evolutionary method focusses on adapting network topology and complexifying a definable structure. For the SQuARC framework, a more generic implementation is desired: an evolutionary search algorithm that uses co-evolution and speciation, but features a minimalistic implementation not based on any specific structure or representation. For these reasons, an adaptation of the steady-state Microbial Genetic Algorithm (MGA) [76] combined with novelty search is used here. The MGA is a genetic algorithm reduced to its basics, featuring horizontal gene transfer (through bacterial conjugation) and asynchronous changes in population where individuals can survive long periods.

To apply the MGA to the problem a number of adaptations are required. Caching fitness values in the standard steady-state fashion is not possible (as in Chapter 7), as fitness is relative to other solutions found and stored in the growing archive. In this implementation, no individual fitnesses are stored across generations, however the same steady-state population dynamics are kept, i.e. individuals are not culled, and may persist across many generations.

An overview of the evolutionary loop is given in Fig. 8.4. The complete process is also outlined in pseudo-code in Algorithm 3.

At the beginning of the search process, a random population is created. In the population, both the substrate configurations and the resulting behaviours *B* are stored. This initial population is then added to the archive *A* and database *D*.

At step 1, tournament selection with a tournament size of two is used. To ensure speciation, the first parent is picked at random and the second is chosen within some proximity to the other determined by the MGA parameter *deme size*. In this step, the fitnesses (novelty) of both behaviours are calculated relative to population *P* and archive *A*. The individual with the larger distance, that is, occupying the less dense region of the behaviour space, is adjudged the winner. This elicits the selection pressure towards novel

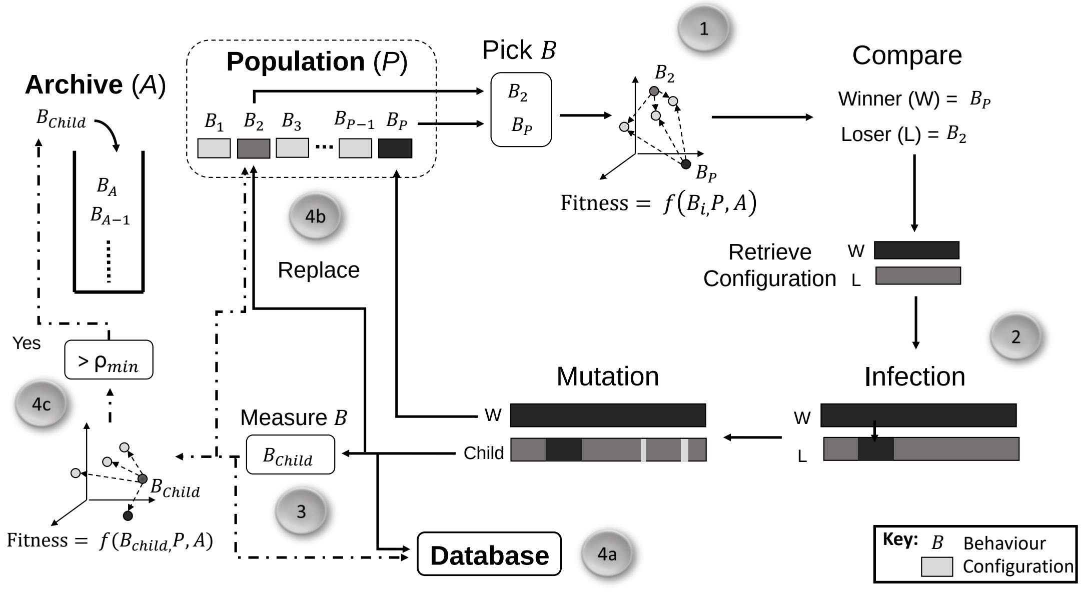


Figure 8.4: Adapted microbial GA with novelty search.

solutions. The microbial GA differs from other conventional GAs as the weaker (here, less novel) individual becomes "infected" by the stronger (more novel) one, replacing its original self in the population.

At step 2, the configurations of both behaviours are retrieved and manipulated. This constitutes the infection and mutation phase. In the infection phase, the weaker parent undergoes horizontal gene transfer becoming a percentage of the winner and loser. The genetic information of the weaker parent does not disappear in this process, as some percentage defined by the recombination rate parameter remains intact. In the mutation phase, the weaker parent undergoes multiple point-mutations, becoming the new offspring.

At step 3, the configuration of the new offspring is untested, therefore the behaviour B_{Child} of the individual needs to be updated. At steps 4*a* and 4*b*, the offspring's behaviour and configuration are added to the database *D* and it replaces the loser in the population *P*.

At the last step 4*c*, the fitness/novelty of the offspring B_{Child} is compared to both the current population *P* and archive *A*. If the novelty of the offspring exceeds the novelty threshold ρ_{min}, the behaviour B_{Child} (configuration is not needed) is added to the archive *A*.

Algorithm 3 Novelty search with microbial GA algorithm

```
$pop \leftarrow random$                                  ▷ initial random population list length $P$
$A \leftarrow pop$                                       ▷ archive initialised
$D \leftarrow pop$                                       ▷ database initialised
while searching do
    $i :\in 1..PopSize$                                  ▷ parent 1 from pop
    $j :\in$ deme $i$                                    ▷ parent 2 from deme
    if $f(pop(i), A, pop) > f(pop(j), A, pop)$ then
        $winner, loser \leftarrow i, j$                  ▷ fitness is novelty
    else
        $winner, loser \leftarrow j, i$
    $child \leftarrow infection(winner, loser)$
    $child \leftarrow mutation(child)$
    $pop(loser) \leftarrow child$
    if $child$ is sufficiently novel then
        add $child$ to $A$
    add $child$ to $D$
    if generation == $n \times update_{gen}$ then
        update novelty threshold $\rho_{min}$
```

Overall, three fitness values are calculated at each generation. Two fitness evaluations occur in the selection phase and a third fitness evaluation is carried out on the offspring, in order to update the archive. The computational complexity of the fitness function is $O(nd + kn)$ using an exhaustive k-nearest neighbour search. As the dimension d of the archive/behaviour space is small ($d = 3$ property measures in the later example), the number of k-neighbours (here $k = 15$) has the dominant effect. This value of k is chosen experimentally; larger k-values improve accuracy but increase complexity. As the archive size increases, complexity increases proportional to archive size n. To reduce complexity, Lehman and Stanley [110] describe a method to bound the archive using a limited stack size. They find that removing the earliest explored behaviours, which also results in some backtracking, does not significantly harm exploration performance in all cases.

8.5 Applied to Echo State Networks

In this section, we begin by defining the behaviour space of interest and characterising the quality of the reference substrate. The chosen reference substrate is the *virtual* ESN substrate.

As a basic demonstration of the framework, a three-dimensional space is chosen using properties/metrics described in section 8.3: memory capacity (*MC*), kernel quality rank (*KR*) and generalisation rank (*GR*). These three measures capture different aspects of the reservoir, both chaos and order, and are simple to apply to physical systems.

To evaluate and validate the framework, multiple ESN network sizes are evolved and assessed against a control comprising of random populations the same size as *D*. The four ESN network sizes chosen are: 25, 50, 100, and 200 node. This provides a small spectrum to assess the framework with, from simple to more complicated reservoirs.

If novelty search performs well, i.e. better than random, with each ESN network size, it might be possible to extrapolate that the same will hold true of other network sizes, and possibly different substrates.

8.5.1 Measuring Quality

To evaluate the quality of each ESN network size, a simple metric (Eqns. (8.14) and (8.15)) measures how much of the behaviour space is covered: greater coverage implies a greater degree of dynamical freedom. Statistical measures of dispersion such as standard deviation, variance, mean absolute deviation and inter-quartile range are not particularly suitable: they downplay outliers, whereas we want to push the boundaries of the region explored. Instead, the behaviour space is divided into discrete volumes, representing these ‘behaviour voxels’, and the quality measure defined counts how many ‘behaviour voxels’ are occupied: the more such behaviours, the larger the volume of space explored.

In the $3d$ example, this discretised behaviour space is captured by a cube represented by the $3d$ array $B_{i,j,k}$; where the coordinate j captures discretised memory capacity values (a continuous-values measure) and i and k capture the kernel and generalisation rank values (already discrete); the assignment of metrics to specific coordinates is arbitrary and does not affect exploration. The appropriate size of B can be deduced from measurement constraints and from the MC bounds imposed on ESNs: $KR \leq N, GR \leq N$ and $MC \leq N$. A simple discretisation of MC is chosen, taking the integer part, so B is an $N \times N \times N$ array.

B is defined as follows. $B_{i,j,k} = 1$ (occupied) if there is at least one individual in the database D with kernel rank $KR = i$, discretised memory capacity $MC = j$ and generalisation rank $GR = k$; otherwise $B_{i,j,k} = 0$ (unoccupied):

$$B_{i,j,k} \leftarrow \exists d \in D \text{ s.t. } KR(d) = i \wedge \lfloor MC(d) \rfloor = j \wedge GR(d) = k \tag{8.14}$$

The total space covered, ϕ, in an evolutionary run is the number of occupied points in $B_{i,j,k}$; $0 \leq \phi \leq N^3$:

$$\phi = \sum_{i=0}^{N} \sum_{j=0}^{N} \sum_{k=0}^{N} B_{i,j,k} \tag{8.15}$$

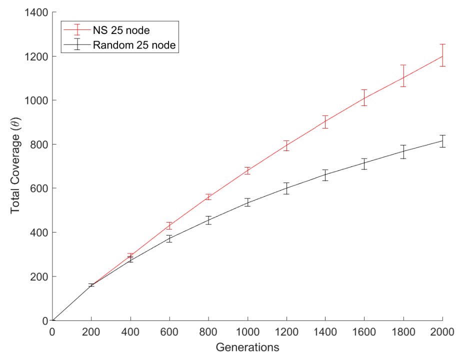
The behavioural voxel occupancy measure works well for this $3d$ case. For higher dimensional behaviour spaces (ones based on more properties/metrics) where the total number of voxels is considerably larger, the representation and measure may need adapting.

8.5.2 Experimental Parameters

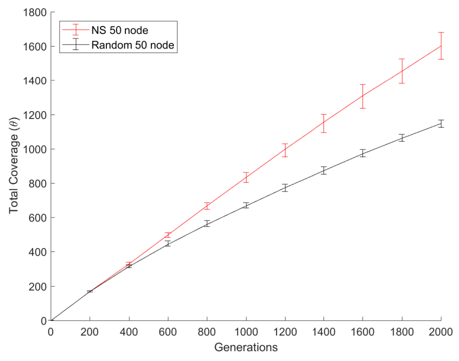
In the following experiments, regardless of ESN network size etc., the same restrictions are placed on global parameter ranges and local weights, and the same weight initiation processes is applied. For example, global parameters ranges include: an internal weight (W) scaling between [0, 2], input scaling [-1, 1], leak rate [0, 1], and the sparseness of W [0,1]. For both random and novelty search, at creation a reservoir has each global parameter drawn from a uniform random distribution, as well as input weights and internal weights drawn uniformly from other ranges; W_{in} between [-1, 1] and W between [-0.5, 0.5].

For the evolutionary algorithm, the following GA parameters are selected from preliminary experiments: *population size* = 200, *deme* = 40, *recombination rate* = 1, *mutation rate* = 0.2, $\rho_{min} = 3$, and ρ_{min} *update* = 200 generations.

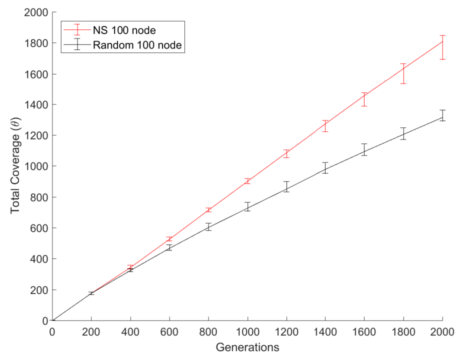
To compare novelty search and the random control, 10 runs are conducted, with a limit of 2000 generations for novelty search, and 2000 randomly initialised reservoirs for random search.



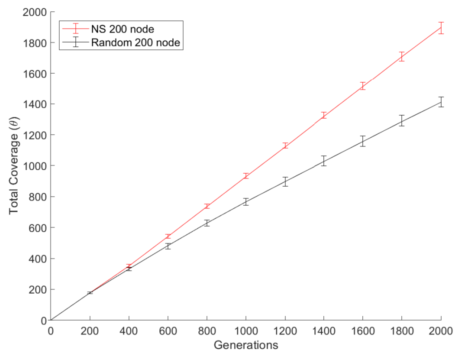
(a) 25 node



(b) 50 node



(c) 100 node



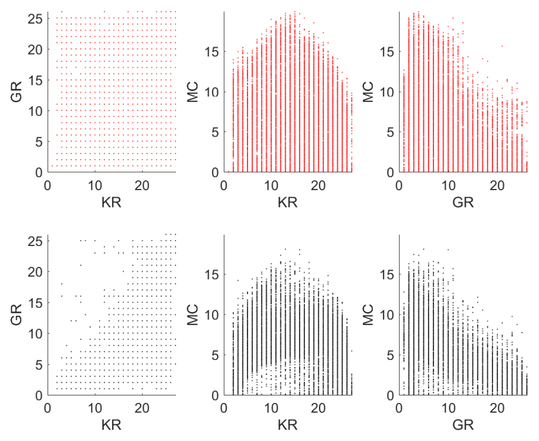
(d) 200 node

Figure 8.5: Average coverage (over 10 runs) of behaviour space against number of generations. Error bars show minimum and maximum coverage.

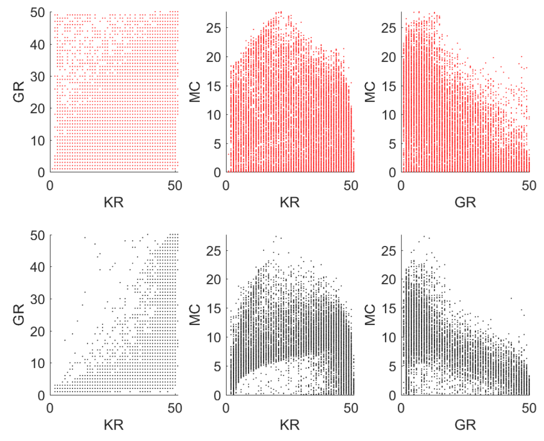
8.5.3 Results: Random Versus Novelty Search

For every ESN network size, novelty search is able to explore a greater area of the behaviour space than the control (random search) in the same time. The total coverage (θ) of the behaviour space versus generations (or database size) is shown in Fig. 8.5. The results show that, with more generations novelty search can continue to explore an even greater area than random search, increasing linearly with the number of generations.

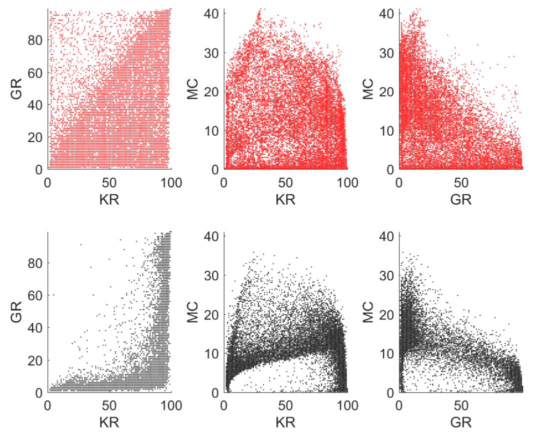
Fig. 8.6 shows all 10 runs of each network size plotted in the behaviour space, helping visualise the difference in coverage. Random search appears to produce similar patterns in the behaviour space with different network sizes. These patterns include sparse regions that are difficult to occupy when uniformly sampling the parameter space, highlighting



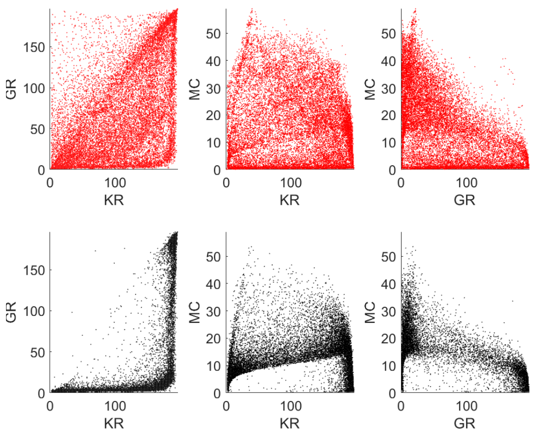
(a) 25 node



(b) 50 node



(c) 100 node



(d) 200 node

Figure 8.6: Plot of all behaviours found through novelty search (Red, Top row) and all behaviours found through random search (Black, Bottom row).

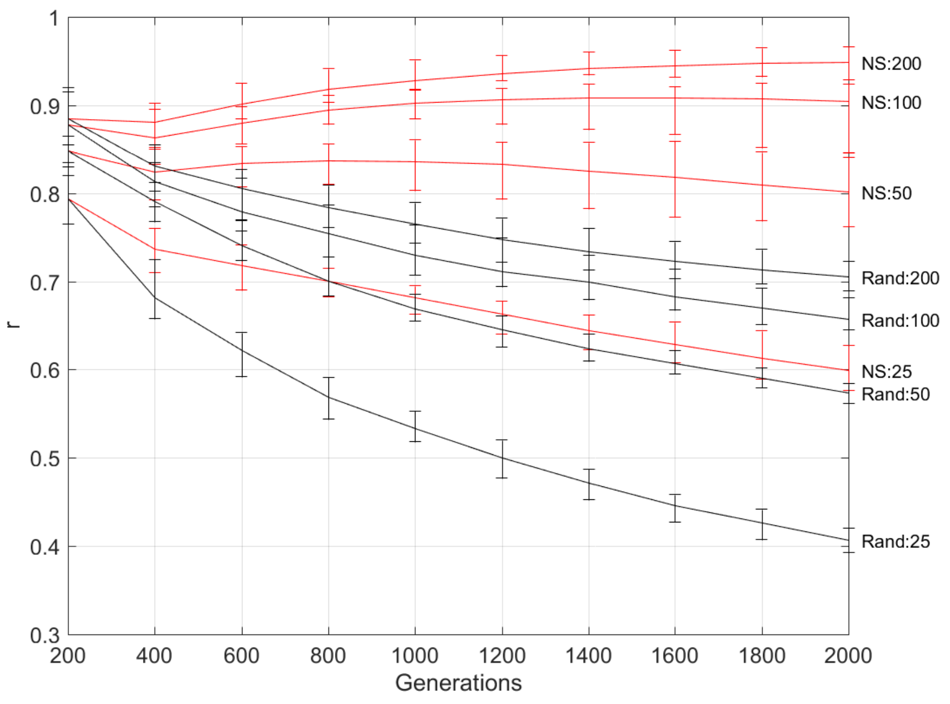


Figure 8.7: A comparison of all experiments shown in fig. 8.5 when plotted as the rate ($r = \theta/generation$).

how deceptive the behaviour space is compared to the parameter space. Novelty search on the other hand covers the behaviour space more uniformly, both filling sparse regions and expanding beyond the region covered by random search. The difference in coverage between the two methods also becomes more distinct with an increase in network size.

Looking at Fig. 8.5 there appears to be a similar linear relationship between coverage (new behaviours) and generations across all ESN network sizes. However, the scales differ, suggesting network size may affect coverage in different ways, e.g. the rate of coverage between 100 and 200 nodes does not increase as much between 25 and 50 nodes. To better understand this, each experiment is replotted using the rate r of total coverage θ per 200 generations; the number of generations coverage was recorded and calculated. For example, a rate $r = 1$ would indicate a new behaviour was found every generation, and a rate $r = 0$ would show exploration halted. In Fig. 8.7, we see that smaller networks find fewer novel solutions per generation and larger networks maintain or increase slightly. This is most likely a product of larger networks having more adjustable parameters (weights). This leads to a higher dynamical degrees of freedom and therefore more distinct behaviours can be found per generation, resulting in smaller networks converging faster to $r = 0$.

8.5.4 Discovering Parameter Patterns

In the ESN literature, intensive work has been carried out to characterise parameter relationships with performance and to provide practical tips for setting parameters to ensure certain dynamics and properties exist, e.g. ensuring the echo state property [86, 117]. However, physical substrates are open systems, can vary significantly, and tend to be largely uncharacterised with little known about parameter–performance relationships. If the same effort was required for every new substrate, there would be little time to freely explore different substrate designs and configuration types.

In this section, we demonstrate how the behaviour space and search for novelty could (re)discover general parameter "rules", distributions and patterns for *free* during the search process.

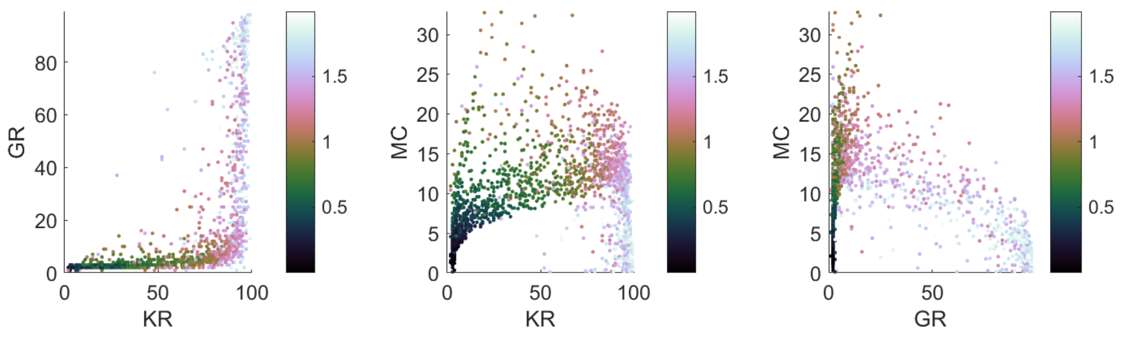
Weight matrix scaling

The first global ESN parameter of interest is internal weight matrix W scaling. This parameter is used in conjunction with the spectral radius. Typically a random sparse W is generated; the spectral radius $\rho(W)$ is computed; and then W is divided by $\rho(W)$. This matrix is then scaled by the W scaling parameter.

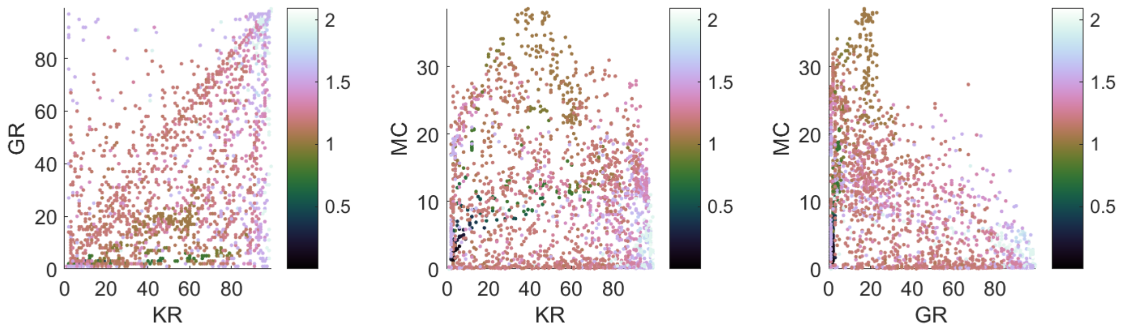
An ESN with a large $\rho(W)$ is said to have chaotic properties and an $\rho(W) < 1$ tends to ensure the echo state property [117].

In this work, the spectral radius is not computed for each reservoir and therefore W is not divided by $\rho(W)$, then scaled. Computing the spectral radius can be computationally expensive, computed as the maximum absolute eigenvalue of the W matrix. Instead, the internal weights W are just globally scaled directly by the parameter.

When plotting the relationship between this parameter and the behaviour space in Fig. 8.8a using random ESNs of 100 nodes, the phase transition between order and chaos can be seen. The W scaling parameter itself pushes the activations (*tanh* functions) in the



(a) Random Search



(b) Novelty Search

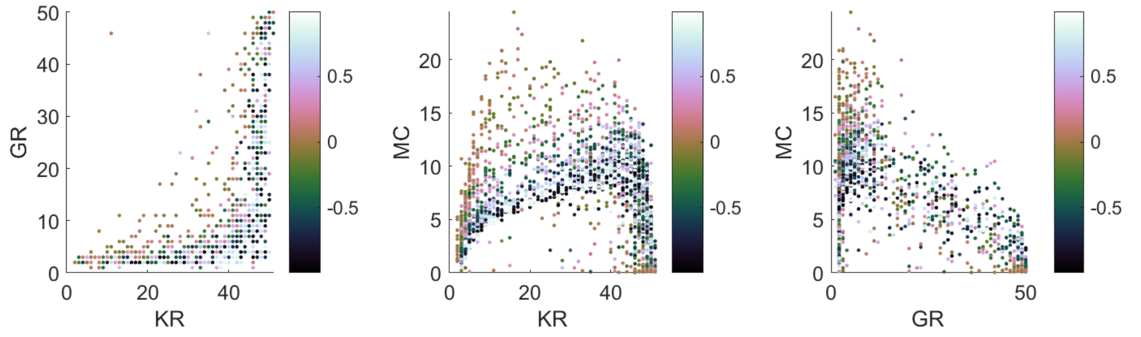
Figure 8.8: *W* scaling: 100 node ESNs.

network towards linear or non-linear characteristics. This affects how fast information from the input degrades in the reservoir with time, thus general stability of the reservoir.

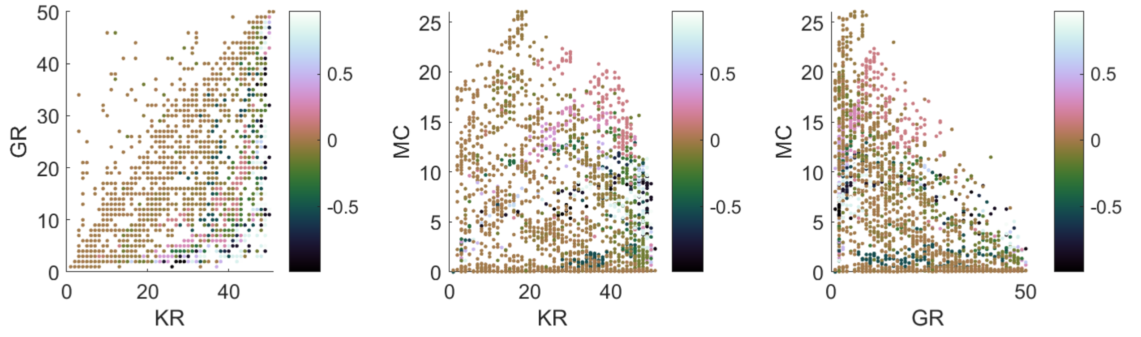
Although with random search (Fig. 8.8a) the phase transition can be clearly seen, with novelty search this appears to break down (Fig. 8.8b). This is arguably because individual weight mutations in the evolutionary loop can make global parameters more redundant. Despite this, there are some relationships still present, for example close to 1, the largest MCs remain and the best trade-offs ($\uparrow$MC = high MC, $\uparrow$KR = high KR and $\downarrow$GR = low GR) are still present.

Input scaling

The next global parameter of interest is input scaling (IS). The input weight matrix W_{in} is usually fully connected and weights are drawn from a uniform random distribution between $[-1, 1]$. The scaling parameter is then selected from the same range. Input scaling determines the non-linearity of the reservoir response. Thus, whether the activation



(a) Random Search



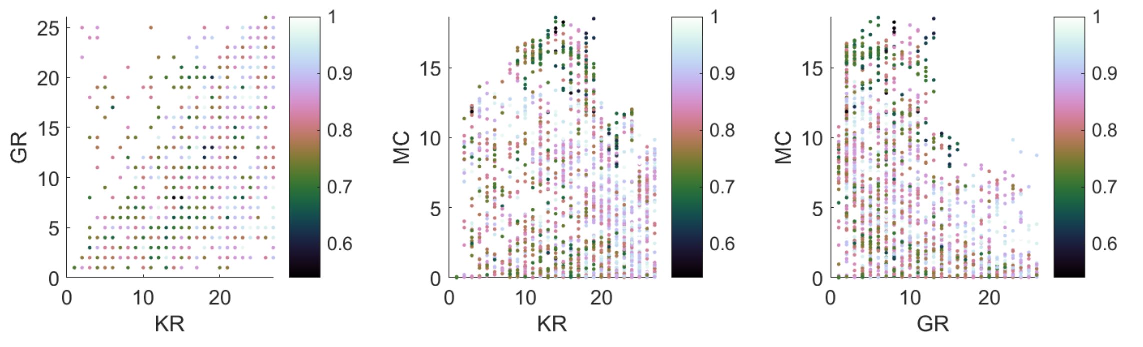
(b) Novelty Search

Figure 8.9: Input Scaling: 50 node ESN.

function operates in a linear regime or a non-linear binary switching regime.

In Fig. 8.9a with 50 node random ESNs, a low input scaling can result in the two extremes, both linear and non-linear ESNs. On the KR vs. MC metric plot, both ↑MC and ↑KR are seen in clusters at either end of the plot, with nothing really present in between.

This changes when moving to novelty search (Fig. 8.9b). A low input scaling occupies most of the behaviour space, however, some clearer patterns also emerge. For example, $0.5 > IS > 0$ gives ↑MC, ↑KR and ↓GR, typically meaning a good trade-off is present. In addition, $0 > IS > -0.5$ gives ↓MC, ↑KR and ↑GR, implying more chaotic reservoirs are present. This implies a low input scaling close to 0 will work well for many tasks, but for difficult tasks desiring an optimal trade-off between memory and non-linearity a small positive (between [0, 0.5]) input scaling may work best.


(a) 25 node

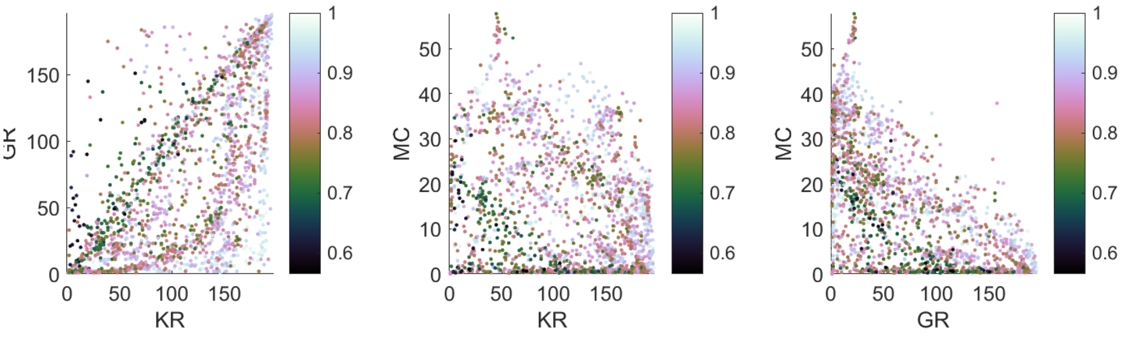

(b) 200 node

Figure 8.10: W connectivity: 25 and 200 node ESNs (novelty search only).

Sparsity/connectivity of the weight matrix

Another global parameter is the sparsity/connectivity of the W matrix. In [117], sparsity is described as having a small effect on performance, however, it is typically recommended, as it can speed up the reservoir update process.

In the random search ESN experiments, the sparsity/connectivity of the ESNs is selected from a uniform random distribution and the parameter is indeed seen to have little effect w.r.t. the behaviour space (see Appendix, Fig. B.4).

When looking at the novelty search results, some small relationships appear to be present. For example, in Fig 8.10 and Fig B.8, a higher connectivity ascending to 1 (paler colours) seems to be preferred to fill the behaviour space as ESN size increases. It is also seen that some relationships reverse as ESN size increases. For example, a low connectivity produces a ↑MC, ↑KR and ↓GR at 25 nodes (a good trade-off) and the same low connectivity results in the opposite at 200 nodes; ↓MC, ↑KR and ↑GR, i.e. generally more chaotic. This still requires further investigation, much like the rest of the parameter

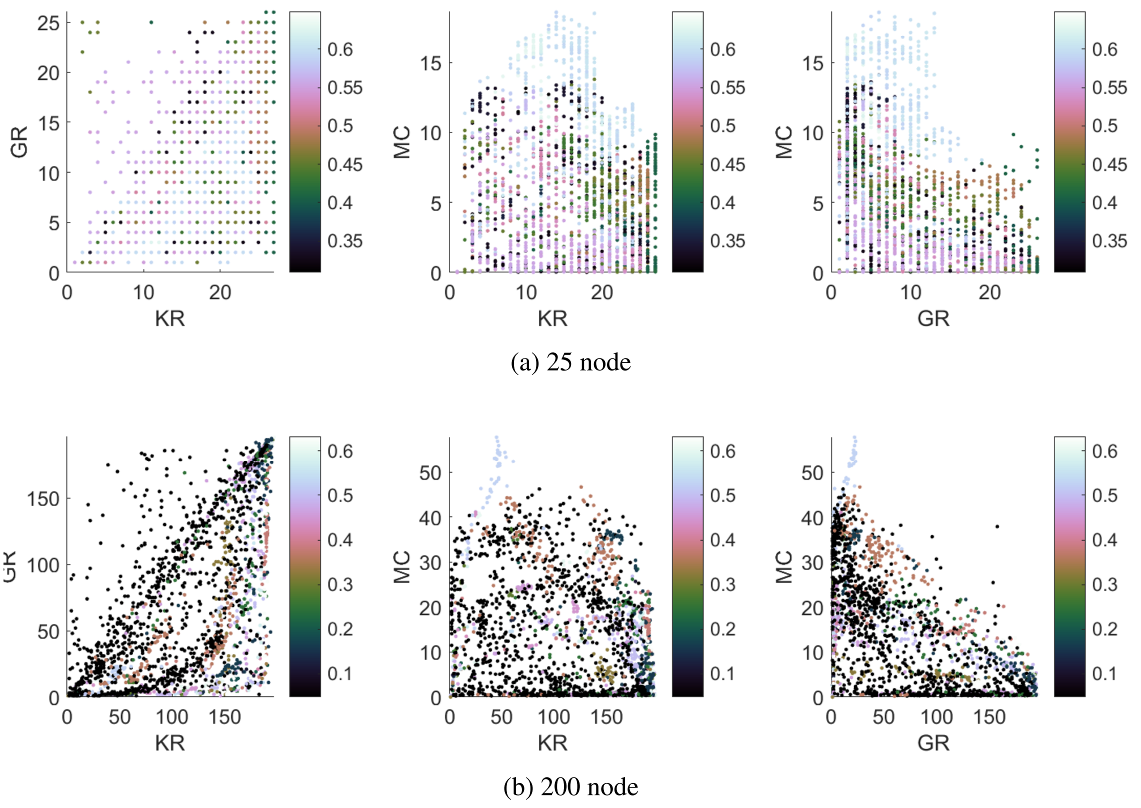


Figure 8.11: Input connectivity: 25 and 200 node ESNs (novelty search only).

plots. However if anything can be learned from this, novelty search may have found a new parameter rule that can guide practitioners when designing ESNs with desired behavioural properties.

Density of the input weight matrix

The next parameter, usually less prioritised with ESNs, is the density (or non-zero elements) of the input weight matrix W_{in}. In general, this is typically set as dense, i.e. one-to-one connectivity between inputs and ESN nodes with no zero elements. In this work, at creation, the W_{in} weights are sampled from a uniform distribution between [-1, 1].

With the novelty search algorithm, mutation can flip an input weight to zero. This results in different input weight densities. With random search, the density is always close to 1. In general, it is found that a low W_{in} connectivity is preferred as ESN size increases (see Fig. 8.11). This is the reverse of the W connectivity pattern mentioned before, suggesting an interesting relationship between the two w.r.t. ESN size and behaviour space coverage. For example, for the 25 node ESN, a $\uparrow W_{in}$ combined with $\downarrow W$ produces the best trade-off. At 200 nodes, $\downarrow W_{in}$ combined with $\uparrow W$ produces a similar trade-off but also occupies other areas of the behaviour space too.

Leak rate

The last global parameter under evolutionary control is leak rate. The leak rate parameter effectively works as a exponential (low-pass) filter on the node state, attempting to reduce the mismatch between input dynamics and reservoir dynamics. Setting a small leak rate is said to slow the dynamics of the reservoir and increase the duration of the short-term memory [117].

The relationship to behaviour space appears to be more distinct with smaller ESNs (see Appendix, Fig. B.3 and B.7), however, determining anything with these current metrics is difficult. In Chapter 4, it is demonstrated that leak rate can have a positive effect on performance. This would suggest the current metrics may not fully capture the impact of adjusting the leak rate parameter.

Conclusion

In each example presented, there is a clear non-linear relationship between the behaviour space and the ESN parameter space, with some behaviours close by in behaviour space often having very different parameter values. This itself shows how difficult it is to select parameters for desirable behaviours, even when the system is closed and well-characterised.

Overall, in this section some general principles of building ESNs have been rediscovered, e.g. W scaling ≈ 1 works well, but other relationships are also observed, e.g. how connectivity of W and W_{in} may have a greater effect as ESN size increases. The potential for novelty search to (re)discover interesting relationships requires further investigation. However, if it *can* find any, or the behaviour space representation can help discover such relationships, the framework will potentially help guide substrate design or configuration practices.

8.6 Representation and Predicting Performance

At this stage, we have described how to define the behaviour space, how to explore/map it and how to use coverage (ϕ) as measure of substrate "quality". However, so far, little intuition is given on how well the behaviour space represents computation in substrates and what relationship properties have to task performance.

Under Abstraction/Representation theory proposed in [81], if the reservoir representation provides a faithful abstract representation of the substrate, it should be possible to provide a prediction of how the system states will evolve. In the context of this framework, we expand the A/R conjecture and hypothesise that if the measures of behavioural properties are substrate-independent and if relationships between properties and task performance can be learned, then it should be possible to predict the performance of one substrate based on another substrate that exhibits similar behaviour.

To assess whether the property measures represent a faithful representation, here we attempt to learn and model the relationship between properties and task performance. How well this relationship can be learned will indicate how accurately the properties represent computation within the substrate.

8.6.1 Prediction Tasks

Determining the property–performance relationship across all tasks is non-trivial. However, relationships between individual tasks and properties are sometimes simple. To predict performance, four benchmark tasks are selected based on dissimilar requirements of reservoir properties: the common non-linear autoregressive moving average (NARMA) task with a 10-th and a 30-th order time-lag; the Santa Fe laser time-series prediction task; and the non-linear channel equalisation (NCE).

The NARMA task was previously used in Chapters 3 and 7. It evaluates a reservoir's ability to model an n-th order highly non-linear dynamical system where the system state depends on the driving input and state history. The laser time-series prediction task was also previously used in Chapters 4 and 7. It predicts the next value of the Santa Fe time-series Competition Data (dataset A)[1].

[1]Dataset available at UCI Machine Learning Repository [200]

The Non-linear Channel Equalisation task introduced in [90] has benchmarked both simulated and physical reservoir systems [153]. The task reconstructs the original *i.i.d* signal $d(n)$ of a noisy non-linear wireless communication channel, given the output $u(n)$ of the channel. To construct reservoir input $u(n)$ (see Eqn. 8.17) $d(n)$ is randomly generated from $-3,-1,+1,+3$ and placed through Eqn. 8.16:

$$\begin{aligned} q(n) = {} & 0.08d(n+2) - 0.12d(n+1) + d(n) \\ & + 0.18d(n-1) - 0.1d(n-2) \\ & + 0.091d(n-3) - 0.05d(n-4) \\ & + 0.04d(n-5) + 0.03d(n-6) + 0.01d(n-7) \end{aligned} \tag{8.16}$$

$$u(n) = q(n) + 0.036q(n)^2 - 0.011q(n)^3 \tag{8.17}$$

Following [90], the input $u(n)$ signal is shifted +30 and the desired task output is $d(t-2)$.

8.6.2 Experimental Set-up

As part of the framework's evaluation phase, the database is assessed on tasks providing a target dataset.

In preliminary experiments (see Appendix E), two machine learning models were investigated and compared, using different feature and data preprocessing settings. From these basic experiments, it was concluded that a 100-neuron feed-forward neural network (FFNN) was the better choice to accurately learn the property-performance relationships.

To model the relationships the FFNN is configured for regression, i.e., given behaviours from the novelty search database, predict performance on a task. The inputs to the FFNN are: MC (continuous-valued), KR and GR (discrete values). The output of the network is task performance (continuous-valued), recorded as the normalised mean squared error (NMSE) of the reservoir with the corresponding input behaviour.

To train the FFNN, Bayesian regularisation is used for 1000 epochs, with the training dataset set as 70% of the data (database D), and 30% set aside for testing.

8.6.3 Prediction Results

In this section multiple FFNNs are trained, per task and per ESN size.

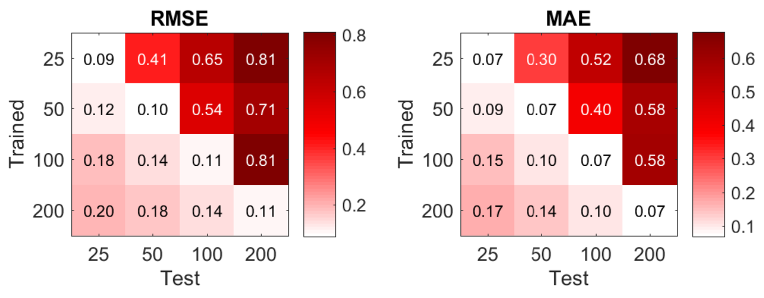
If the property measures are accurate and provide a faithful representation of the substrate, the prediction error of all trained models should be low and similar to each other, i.e. shows a relationship is present, not too difficult to model and holds when the substrate is changed (in this case a different size). However, there will be some deviation in error between models trained with databases holding different behaviours, because having a greater or smaller behavioural range can result in an increase or decrease in complexity of the modelled relationships. For example, reservoirs in the behaviour space around $KR = GR = MC \leq 25$ tend to have similar poor performances on the NARMA-30 task because they do not meet the minimum requirement ($MC \geq 30$). This means the NARMA-30 task is easier to model with the database created by the 25 node ESNs. When databases with larger ESNs are used to model the relationship, prediction error will likely increase.

This is not always true, for some tasks to accurately model the relationship requires a greater variety of behaviours than smaller ESNs can provide (e.g. for the non-linear channel equalisation task). Therefore, an FFNN trained on a database provided by the 200 node ESNs will perform better than one provided by the smaller 25 node ESNs. This is one example of where the non-trivial problem of relating properties to performance presents itself.

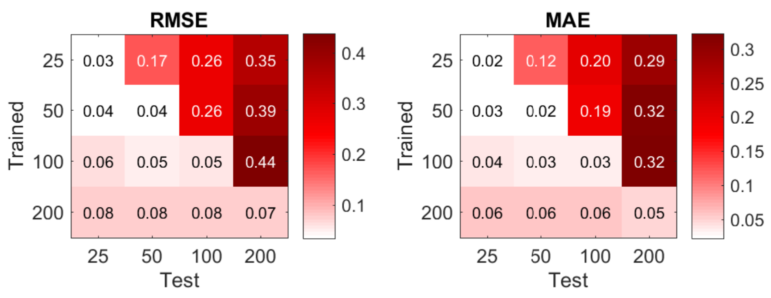
In Fig. 8.12, the results of FFNNs (four per task) trained on different tasks and databases are shown as heat maps. The test error (either root mean square error *RMSE* or mean absolute error *MAE*) of an FFNN trained and tested using the same database is given along the diagonal, where train ESN size y = test ESN size x. When focusing on these results, we tend to see a small deviation between model accuracies, however there are some exceptions, as previously discussed.

Also shown in Fig. 8.12 is how well each FFNN predicts the task performance of reservoirs from different databases, i.e. when train ESN size $y \neq$ test ESN size x. The generalisation to other databases, holding possibly new behaviours, varies depending again on the task. In general, if train $y \geq$ test x, the FFNNs prediction error is usually greater than the original, but considerably less than train $y \leq$ test x .

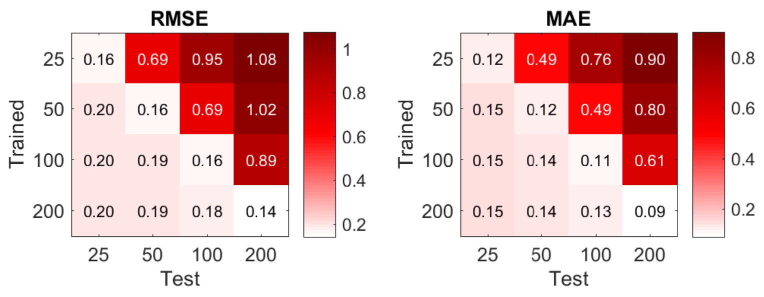
The most likely cause of the first case, when train $y \geq$ test x, is a loss in granularity of



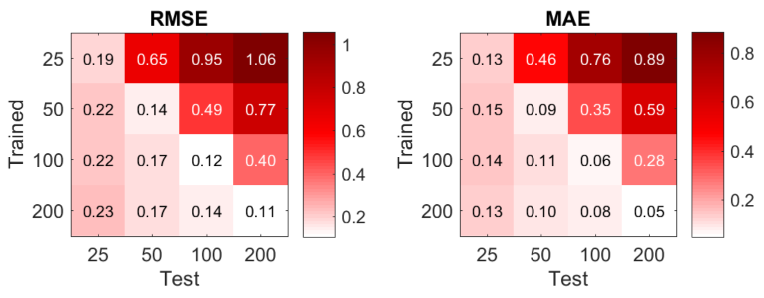
(a) NARMA-10



(b) NARMA-30



(c) Laser



(d) Non-linear Channel Equalisation

Figure 8.12: Mean performance of 20 FFNNs trained and tested on each database (different sized ESN), per task .

the learned relationships between databases. Stretching the same number of data-points to a larger area will result in fewer examples to learn from in each region of the behaviour space, i.e., a loss in resolution that might be important to the learned problem. This can be thought of as a sampling bias or quantisation effect. The second case is simply a result of new behaviours predicted based on no prior data, resulting in a poor prediction.

Overall, the prediction accuracies appear consistent, with small deviations as the behaviour range increases or decreases. This would suggest the behaviour space and property measures are a somewhat faithful representation of the substrate's computational capabilities.

8.7 Completing the Framework

At this stage, the framework definition is almost complete. The hypothesis so far is that the behaviour space of the substrate can be explored and quality can be measured with a faithful representation of the substrates computational mechanisms.

The final part of the framework is: i) to apply each level to a new uncharacterised substrate, ii) to evaluate quality w.r.t. the reference substrate, iii) to use the predictive network from section 8.6 to predict task performance, and iv) to use the prediction to validate the substrate-independence of the behaviour space.

The substrate that is used to demonstrate this part of the framework is the physical substrate investigated in [40, 41, 42] and throughout this thesis. The substrate comprises of a carbon nanotube–polymer composite, forming random networks of semi-conducting nanotubes suspended in a insulating polymer, deposited onto a micro-electrode array.

In earlier chapters (3, 4 and 5), a small amount of characterisation has been done showing that even the best substrate (1% concentration of carbon nanotubes w.r.t. weight mixed with poly-butyl-methacrylate) typically exhibits low memory capacity, despite different methods of configuration. This leads to overall modest performances, but encouraging when compared relative to size, on benchmark tasks such as NARMA-10 and the Santa Fe laser time-series prediction task.

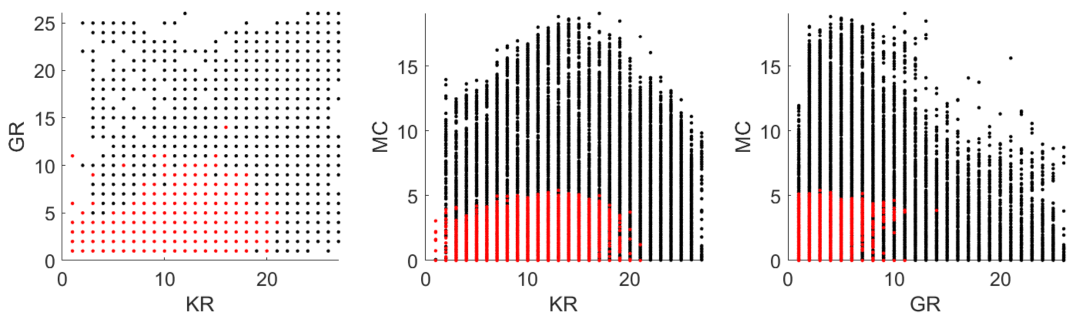
The challenging aspect of characterising this black-box substrate is due to its disordered structure and self-organisation during the fabrication process, making it impractical (or even impossible for the general case) to model internal workings. Originally, the CNT/polymer was proposed as a sandpit substrate to discover whether computer-controlled evolution could exploit a rich source of physical complexity to solve computational problems [18]. Due to its shortcomings, it provides a challenging substrate for the proposed framework and is useful for demonstrating the process.

The training and evaluation of the substrate is conducted on a digital computer. Inputs and representative reservoir states of the substrate are supplied as voltage signals. The adaptable parameters for evolution are the number of input-outputs, input signal gain (equivalent to input weights), a set of static configuration voltages (values and location), and location of any ground connections. Configuration voltages act as local or global biases, perturbing the substrate into a dynamical state that conditions the task input signal. An in-depth description of the how the system is configured is given in Chapter 6.

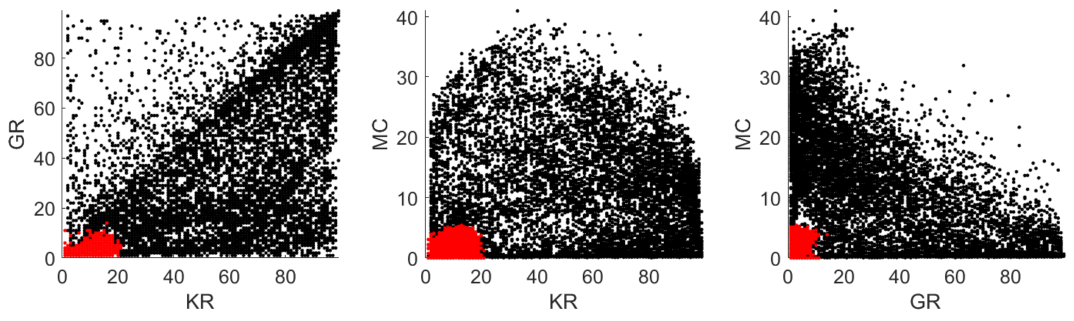
An advantage of physical substrate-based reservoirs is that computational speed of a trained reservoir is limited only by interface hardware and physical response time, which can potentially be all analogue. However, the training process can take considerably longer as multiple runs for statistical tests are often needed. With this substrate in particular, there are many unstable configurations. Therefore, extra evaluations are required to measure statistical stability in order to discard unstable signals from the training process. For this reason alone, predicting performance across substrates and removing the task training process would be decidedly beneficial.

8.7.1 Experimental Parameters

Here, a 1% carbon nanotube poly-butyl-methacrylate (CNT/PBMA) mixture substrate is investigated. The same GA parameters applied to the virtual ESN substrate are reused with the physical substrate. These are: generations limited to 2,000; *population size* = 200; *deme* = 40; *recombination rate* = 1; *mutation rate* = 0.2; $\rho_{min} = 3$; and ρ_{min} *update* = 200 generations. Five runs were conducted here, as the time to train increases significantly.



(a) 25 Node ESN



(b) 100 Node ESN

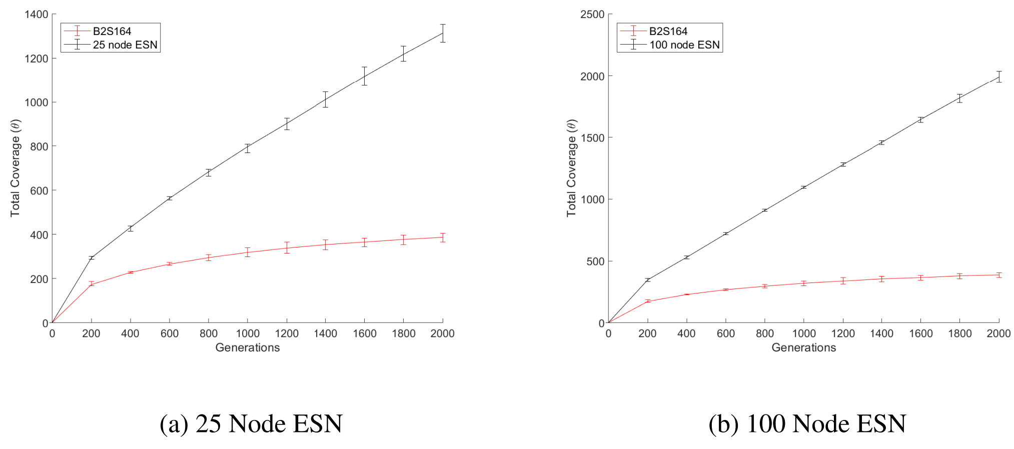
Figure 8.13: Carbon nanotube composite behaviour space (Red) superimposed on ESN behaviour space (Black)

8.7.2 Quality of Physical Substrate

The results of the evolved physical substrate suggest a poorer quality and a limited dynamical degrees-of-freedom compared to the reference ESN substrate. This would appear to be in agreement with previous work comparing the physical substrate to ESNs in Chapters 3, 4 and 5.

When overlaying the explored behaviour space of the physical substrate on top of the reference substrate the difference becomes distinct (see Fig. 8.13). The average coverage of the physical substrate in Fig. 8.14 is almost a magnitude smaller than the 25 node ESN experiment.

The effective number of equivalent nodes an ESN should possess to compare fairly to this physical substrate is unknown. As a rough guide, the physical substrate has up to 64 state observations at any one time, therefore it might be considered generous to compare it to a 25 node ESN, however, this assumes similar non-linear nodes, network dynamics,



(a) 25 Node ESN

(b) 100 Node ESN

Figure 8.14: Comparison of behaviour space covered; carbon nanotubes (Red) and ESNs (Black).

connectivity, etc.

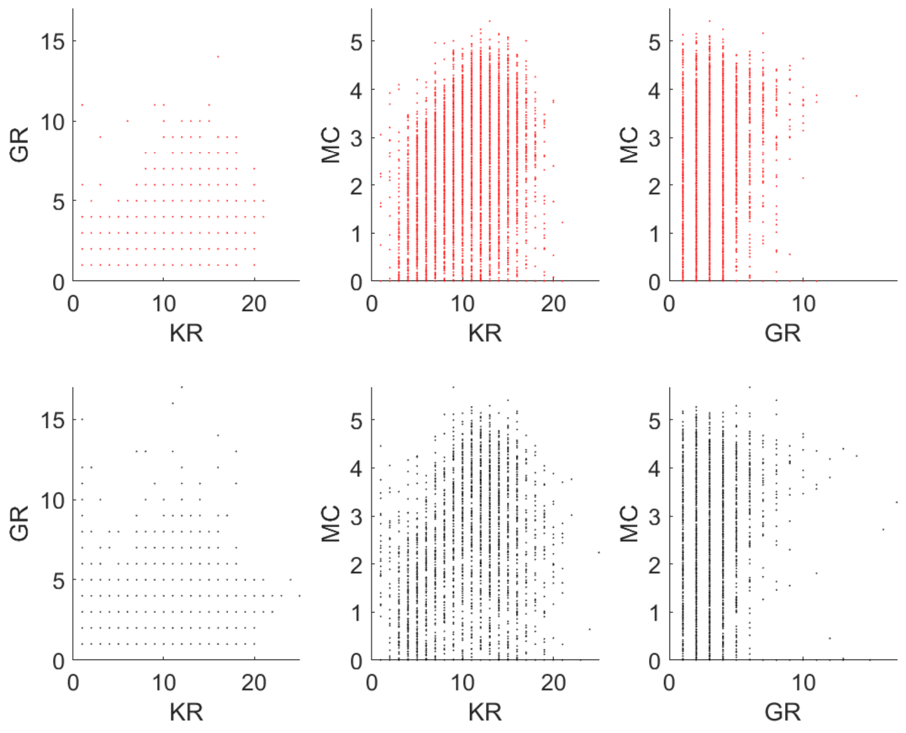
In general, the search struggles to find configurations beyond a memory capacity of 5, reaching what appears to be a memory capacity limit. The bounds on the ranks are also small given only a small number of inputs are typically in use. This would suggest the substrate struggles to exhibit enough (stable) non-linear behaviour to create a strong non-linear projection, and effectively store recent input and state information.

To investigate if the substrate limits are reached, random search is also conducted (Fig. 8.15) and the *novelty rate* is plotted (Fig. 8.16).

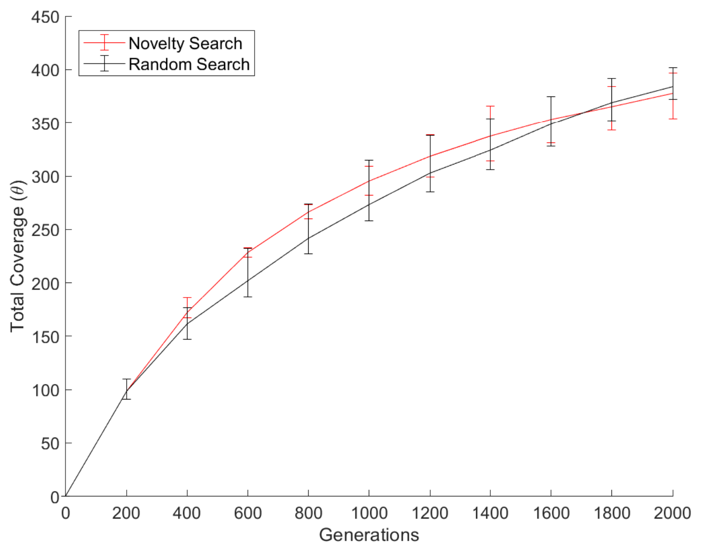
When comparing random search to novelty search in Fig. 8.15b, novelty search on average appears to explore slightly faster and further for roughly 1200 generations but then begins to plateau around $\theta = 360$.

Looking at the novelty rate of the physical substrate (Fig. 8.16), we see that the rate of new behaviours drops considerably over the course of the evolutionary runs. For random search this drops faster, then novelty search reaches the same low novelty rate after 1600 generations, maintaining the same decrease in novelty rate as random.

Fig. 8.16 also shows how the physical substrate compares to a 25-node ESN. The physical substrates rate r starts much lower, around $r = 0.5$ compared to the ESN ($r = 0.8$). This itself begins to suggest the physical substrate has a small degree of dynamical freedom, as less than half of the 200 original random behaviours were unique.



(a) Behaviour space comparison



(b) Total coverage comparison

Figure 8.15: Physical substrate: novelty search (red) versus random search (black).

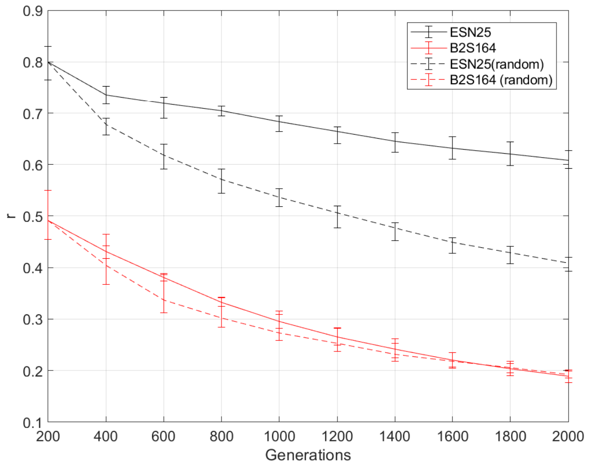


Figure 8.16: The novelty rate ($r = \theta/generation$) of the physical substrate and compared with 25 node ESN.

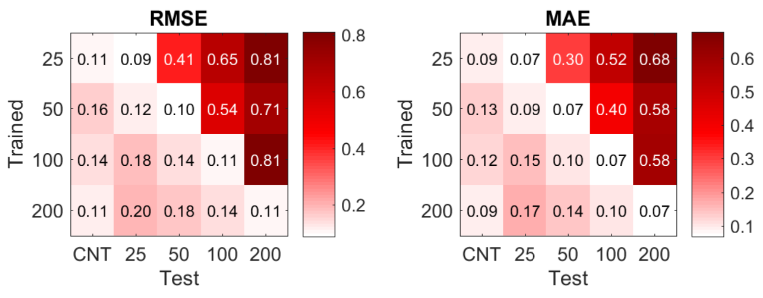
The rate r for both random and novelty search then both decrease at a similar rate, however, this is not the case for the ESN. Although both decrease as the behaviour space is explored/filled, the difference in r gradually increases, with random dropping early and continuing to get smaller; similar divergence patterns are seen at other ESN sizes, see Fig. 8.6.

This small difference in r across the generations, combined with a low starting and very low finishing r, plus a struggle to find unique behaviours (Fig. 8.15a), suggests the physical substrates dynamical limits have been reached. Although this shows the substrates are limited for reservoir computing, it more importantly demonstrates that the framework can approximate and quantify the dynamical boundaries of the substrate.

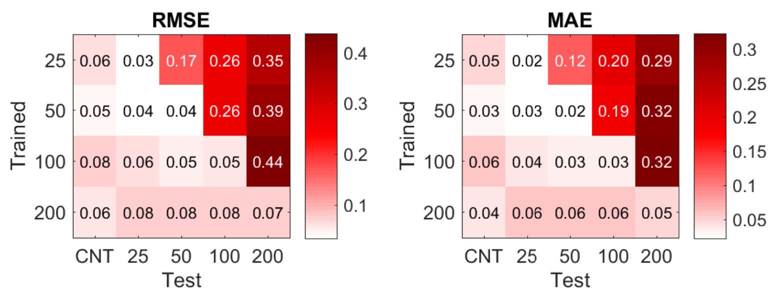
8.7.3 Prediction transfer

In the final part of the framework, the substrate independence of all levels can be evaluated. To do this, all the single-task FFNNs from the learning level (section 8.6) are used to predict the performance of the carbon nanotube composite. Then, using the difference Δ between prediction errors of both substrates we measure how accurately the modelled relationships translate across substrates.

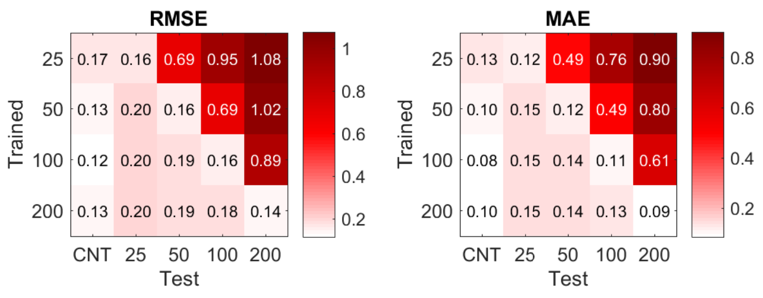
The measure Δ represents the difference between the trained *ESN substrates* test error



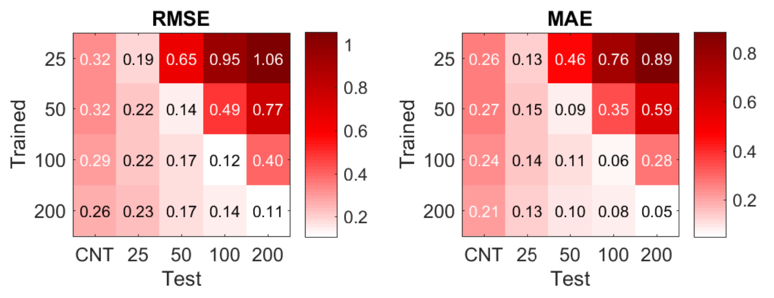
(a) NARMA-10



(b) NARMA-30



(c) Laser



(d) Non-linear Channel Equalisation

Figure 8.17: FFNN prediction errors of all trained and tested ESNs, and tested on physical substrate.

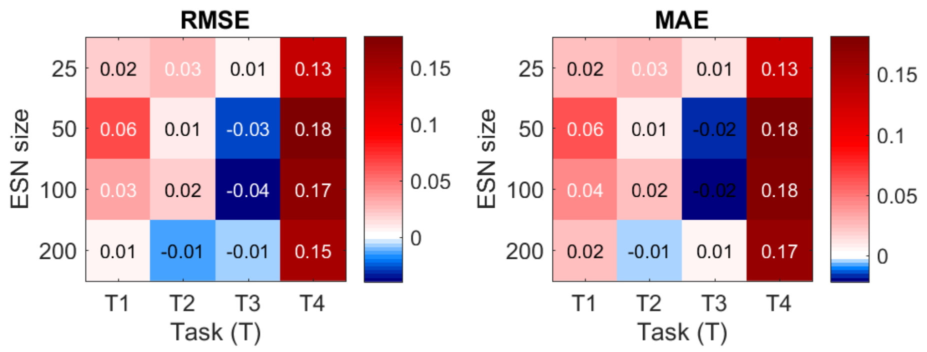


Figure 8.18: The average difference Δ between prediction errors for ESNs trained/tested using same size and physical substrate, across the four tasks: T1 = NARMA-10, T2 = NARMA-30, T3= Laser, T4= Non-linear channel equalisation.).

and new *physical substrates* test error. To support the hypothesis that the behaviour space and the property measures have substrate independence, a low Δ close to zero is desired.

Prior to the prediction tests, one database created from a single run is assessed on the four tasks, creating the test set for the physical substrate. This is done here for evaluation purposes only, if the framework is found to be substrate-independent the task evaluation step on the physical substrate can be ignored.

The prediction error of all FFNNs, trained with different size ESNs and tested on the substrate for all four tasks, are given in Fig. 8.17. The first column in each grid provides the test error of the FFNNs when predicting the physical substrates task performance.

On every task except non-linear channel equalisation, the FFNNs predict the task errors of the physical substrate almost as well as an FFNN trained and tested with the same ESN size. As seen in Fig. 8.17d, the FFNNs trained to predict the non-linear channel equalisation task struggle to accurately translate the learnt relationships to the physical substrate. Although the prediction error is worse than if trained and tested on the same size ESN, the error is considerably lower than if an FFNN was trained using a small ESN size then tested on a larger ESN size.

Another interesting result is that, in some cases, the FFNN trained with the 200 node ESN more accurately predicts the performance of the physical substrate than a FFNN trained with a smaller sized ESN. This is somewhat counter-intuitive because an FFNN trained with a 200 node ESN has fewer examples in the behaviour space occupied by

RMSE ESN size	T1	T2	T3	T4
25	0.02	0.03	0.01	0.13
50	0.06	0.01	-0.03	0.18
100	0.03	0.02	-0.04	0.17
200	0.01	-0.01	-0.01	0.15

MAE ESN size	T1	T2	T3	T4
25	0.02	0.03	0.01	0.13
50	0.06	0.01	-0.02	0.18
100	0.04	0.02	-0.02	0.18
200	0.02	-0.01	0.01	0.17

the physical substrate, i.e. in the region within $KR = GR = MC \leq 25$, thus would likely over-generalise/under-fit this region, as it does when tested with the 25 node ESN.

All these trends are summarised in the Δ plot, Fig. 8.18. The average Δ (in units RMSE or MAE) is given for each task and for all FFNNs (10 per ESN size), applied to the physical substrate. For the first three tasks, small Δ's close to zero are present in most cases, despite ESN size. However, as before, Δ increases significantly for the non-linear channel equalisation task.

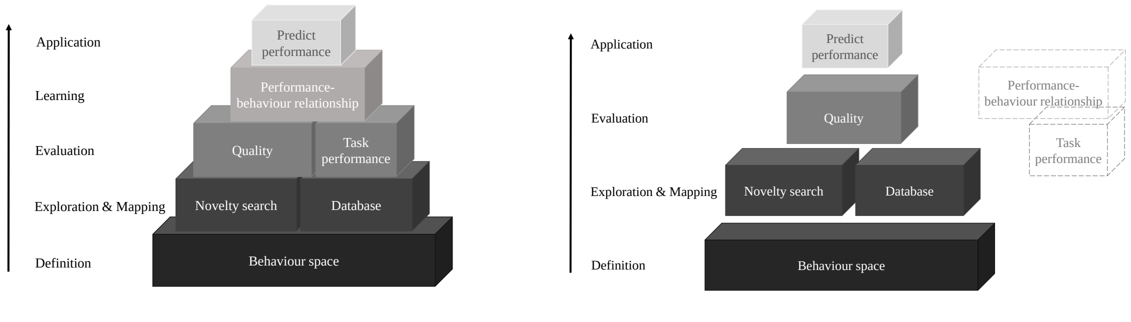
The reasons why the modelled relationships of the non-linear channel equalisation task struggle to translate across substrates is still unknown and requires further investigation. However, a potential lead to explaining this is that the variation in performance across the behaviour space is very low, with reservoirs requiring very low metric values to perform well at this task. Therefore, the greater variation seen with the physical substrate is likely to be poorly modelled by the FFNNs trained with larger ESNs.

Overall, the results in this section suggest the framework has a good level of substrate independence and the behaviour space can accurately represent the substrate's computational properties. It also highlights again the non-trivial nature of the task–property relationship and how some tasks can be more difficult to model, or require extra thought and manipulation, than others.

These last results also show that the learning phase (and task evaluation) only needs to be performed with the reference substrate, once per task, leading to a significant reduction in time to evaluate and test new substrates on specific tasks. For example, one could evaluate the reference substrate's database on a new task, train an FFNN, then predict whether a previously explored physical substrate would be suitable, perform well on the task, or even what the best configuration would be, without having to test directly on the substrate.

8.8 Summary

It has recently been shown that the coexistence of both linear and non-linear dynamics can significantly boost reservoir performance for certain tasks [85]. This relates directly to a fundamental question in reservoir computing: for a given task, what characteristics of



(a) Phase 1 – Reference Substrate (b) Phase 2 – Test Substrate

Figure 8.19: Framework summary. a) Reference substrate mapped providing a reference quality and learned relationships to predict performance. b) Test substrate is assessed for quality. No task assessment is necessary and learning phase is skipped. Performance of test substrate is predicted using learned relationships from reference.

a dynamical system or substrate are crucial for meaningful information processing? The presented framework attempts to tackle this question by focussing on the substrate rather than the specific task. In the process, solutions to two non-trivial problems are proposed; (i) how to characterise the quality of any substrate for reservoir computing, and (ii) how computational properties relate to performance.

To fully utilise the framework, two phases must be completed, see Fig. 8.19. First, the lower levels of the framework are applied to the reference substrate, providing context to future quality measures on other substrates. Then, the upper levels are applied to learn the transferable relationships between substrate behaviours/properties and task performance; helping validate the framework and predict performances of similarly behaved substrates. The second phase applies a reduced set of levels (see Fig. 8.19b) to a new substrate for which a measure of quality is desired. Then, the learned transferable relationships can be used to predict how well the second substrate could perform a task without having to be assessed directly on the task.

Throughout this chapter, each level of the framework has been described and tested. In some cases, adaptations and the removal of different levels/building blocks have also been discussed. One promising adaptation of the framework is provided in Appendix D where a *multi-thread* version of the framework is tested. In another experiment, the importance of the archive is tested (see Appendix C). Interestingly, when the archive is replaced by the database the change in total coverage appears minimal. The only apparent benefit to using the archive, instead of the database, is a computationally cheaper fitness evaluation. This suggests many further improvements are possible at least at the *exploration* & *mapping* level, e.g. a reduction in archive size [110] and the introduction of minimal criteria for novelty [109]. This demonstrates both the flexibility and potential power of the framework where building blocks can be changed, integrating new techniques or measures not currently available.

What has been learnt whilst using the framework is that exploration (novelty search) has a unique ability to help uncover hidden parameter relationships (section 8.5.4) and outline substrate limitations (section 8.7.2). The framework even helps explain why the carbon nanotube composite struggles to compete with ESNs in previous work.

Another feature highlighted is the non-trivial problem of relating properties to task performance (section 8.6.3), suggesting current measures of dynamical properties only provide a partial view of the full characteristics of reservoir systems. Therefore, in order to fully understand these systems we need to explore each property and its relationship to others.

Ultimately the framework has provided a basic methodology to compare and evaluate any substrate for reservoir computing, something not achieved, believed to be possible, or widely discussed in the reservoir computing community. The advantage of having such a framework can only be speculated, but any method to compare systems, or even techniques for perturbing or observing systems, has the chance to rapidly improve the design and implementation process. Furthermore, it potentially opens the reservoir computing field to a wider audience where new interesting unconventional computing substrates could appear.

Chapter 9

Conclusion and Future Work

9.1 Outcomes of Thesis

At the beginning of the thesis a simple question is posed; can the reservoir computing (RC) framework improve aspects of the evolution *in materio* (EiM) technique? To answer this, a new hybrid *reservoir computing in materio* (RCiM) framework has been developed.

To evaluate the new approach, carbon nanotube composites were evolved *in materio* and trained *in silico* to solve a variety of reservoir computing tasks. The results show these techniques complement each other well, adding optimisation to RC and a level of abstraction to EiM (see Chapters 3, 4, and 5). However, each chapter uncovered potential limitations of both techniques, for example, the hardware scaling problem accompanied with computing *in materio*, and a single reservoirs inability to model multi-scale dynamics. Solutions to such problems were given in Chapters 6 and 7.

Perhaps the most fundamental problem to arise from early exploratory work was how to better understand and measure the underlying computational mechanisms being exploited by evolution. This basic, but non-trivial problem, formalised itself as Chapter 8. The SQuARC framework solves this problem by abandoning the idea of characterisation through tasks. Instead, open-ended evolution and search exploration is embraced to characterise a substrate with respect to its dynamical degrees of freedom. Mapping the expressiveness of a substrate (based on dynamical properties, not tasks) defines and quantifies what dynamical traits are being exploited.

This ambitious framework has resulted in a unique synergy of the two fields, demonstrating the potential not only to *assess the quality* of any substrate for reservoir computing – using computer-controlled evolution – but also to *predict the performance* of substrates expressing similar computational properties.

9.1.1 Thesis Chapter Summary

A summary of each experimental chapter and its technical details are given in this section. This provides a useful reference to significant results and outcomes of the thesis.

Preliminary experiments in **Chapter 3** show both the reservoir computing and evolution *in materio* concepts are compatible. It demonstrates that computer-controlled evolution can *pre-train* physical substrates into working reservoir computers using the RCiM framework.

- Materials under-test: three carbon nanotube/polymer composites (PMMA 0.1%, PBMA 0.53%, PBMA 1%) dropcast onto glass slides with 12-electrodes; a gold resistor array patterned onto glass with 16 electrodes.
- Two system settings (*short-circuit* and *open-circuit*) were also tested.
- Benchmark tasks: NARMA-5 and NARMA-10 tasks; wave generator task.
- A $1+\lambda$ evolutionary strategy is used to evolve input-output mappings and control signals (static voltages). Inside the training loop, the output states of the material are used to train the readout layer, reducing the error (NRMSE) between the the reservoir's states and the target signal.
- The encoding of the configuration was represented by a 21-gene genotype, featuring active and non-active genes.
- Results: the material has a significant role in the computing system; when removed (in the *short-* and *open-circuit* setting) the system performs poorly.
- On benchmark tasks, no significant difference is found between the resistor array and materials, suggesting little more than resistivity is present in the materials. This is confirmed when analysing memory capacity, which is found to be low, typically $MC \leq 4$.

- Despite poor performances on the NARMA tasks, the materials performed well on the wave generator task compared to atomic switch networks.
- In the experiment, the difference in nanotube density appears to have a minimal effect.
- The stability of configurations, referred to as solution *drift*, is discussed, as well as the hardware limitations of a small electrode array.

In **Chapter 4**, a more in-depth analysis of the RCiM framework is given, including ways to improve the previous approach in Chapter 3. Four key features are assessed: i) performance compared to random search, ii) a weighted input mechanism directing to multiple electrodes (instead of one), iii) introduction of an output filter to reduce the mismatch between task and material time-scales, and iv) the effect of solution/configuration drift.

- The two system settings (*short-* and *open-circuit*) are dropped; the previous four materials (including the resistor array) are used.
- A similar $1+\lambda$ evolutionary strategy and training process is applied. The genetic encoding is altered slightly to feature input weights and the time-scaling parameter (α).
- Task used to benchmark each feature: the Santa Fe *laser time-series prediction* task.
- The more traditional reservoir computing input mechanism is introduced, moving from a *one-for-one* input mapping to a *one-to-many*. The leak rate parameter is also introduced, adding an extra evolvable parameter that adjusts a *smoothing* filter on the output states. The results of all combinations (e.g. input weighting "on" and time-scaling "off", etc.) show, when combined together, both *poor* and *good* performance reservoirs can be fine tuned.
- The results confirm evolutionary search produces reservoirs with better performances than random. They also show the 1% density SWCNT/PBMA material (near the 1% percolation threshold) tends to outperform the others, on this task.

- For the laser task, the performance difference between carbon nanoutbe materials and the resistor array is significant.
- In the final experiment, the repeatability of performance is measured. This is evaluated by reapplying configurations after multiple time periods (an hour, a day, and a week), and after reconfiguration to other tasks. The results show error can increase with time, but tends to settle after a day, with up to an 11% increase in error.
- The performances achieved in Chapter 4 on the laser task is very competitive in comparison to other virtual and physical reservoir computers,

In **Chapter 5**, the RCiM framework is compared directly to the EiM technique. The highly-programmable RCiM approach is assessed to see whether increased programmability (over the EiM approach) results in a loss in performance. At the end of the chapter, the RCiM approach is also compared to *in silico* reservoirs (of approximately equivalent size) to determine what effect *sub-sampling* of the reservoir states has on reservoir performance. As a result, a greater understanding is gained of what material characteristics are being exploited by evolution.

- Six materials were used: 0.1%, 0.53%, 1%, two 0.71% materials, and the gold resistor array.
- To enable comparison with EiM, a new *non-temporal* benchmark is used: the *Iris* classification task.
- The same training process in Chapters 3 and 4 is used; fitness is determined by the prediction accuracy of discrete classes.
- For the first time, multiple task inputs and outputs are required.
- On average, RCiM produces higher training and test accuracies than EiM. Like Chapter 4, the 1% density material features the best performances.
- Greater diversity in performance (w.r.t to SWCNT density) is observed when compared to Chapter 3. As with Chapter 4, the difference in performance between materials and resistor array is measured to be significant.

- The materials performances compare favourably to the *in silico* evolutionary programming technique, Cartesian Genetic Programming (CGP).

- When measuring the effects of sub-sampling reservoirs *in silico*, a significant difference is found. This result suggests sub-sampling *in materio* systems can have either a detrimental or positive effect on performance, depending on the task. For example, for the classification task, access to more states results in over-fitting. However, in Chapter 4 for the laser task, sub-sampling results in poorer performances.

- No significant difference is found between the performance of the best 1% density material and *in silico* reservoirs (ESNs) with 7-nodes (including the 50-node ESN sampled to form 7-nodes).

- Analysis of the *in silico* parameters explain why both *in materio* and *in silico* reservoirs perform similarly well on this task. The ESNs evolve to feature dynamical traits common to the carbon nanotube materials.

In **Chapter 6** a new hardware platform is developed, increasing the 12 electrode system from previous chapters (3, 4 and 5) to 64 electrodes. During the testing phase, evolution's ability to exploit not only the substrate but also the surrounding system was demonstrated on an early version, leading to a second hardware design with substrate isolation.

- In order to route 64 electrodes to the previous data acquisition system, a new external routing board has been designed. Two iterations of the board have been created, the second improving substrate isolation compared to the first.

- Due to complications in manufacturing glass electrode arrays, a PCB electrode array has been designed and tested.

- New materials samples are used with the new electrode arrays. An outline of the fabrication process is given in section 6.2.3.

- A significant problem highlighted in Chapters 3 and 4 is solution stability. To compensate, a new measurement strategy is used, involving multiple readings and the removal of unstable electrode readings to improve measurement precision.

- A new genetic encoding is defined. This new configuration representation allows both input weights and control signals to be used simultaneously. It is also designed to take into account other hardware restrictions, such as a limit of 32 inputs at any one time.

- When testing the first iteration of the routing board, it was discovered that evolution found ways to exploit other electronic components of the measurement system. This anomaly was highlighted when simple LED arrays produced implausibly impressive performances, featuring up to a 92% improvement compared to previous carbon nanotube experiments in Chapters 3 and 4. This was due to a measurement error, caused by signal "ghosting", achieving greater memory than the substrate could exhibit alone. This was due to poor impedance matching. In the second iteration, ghosting was removed using analogue buffers. Substrate isolation improved, and the implausible performances dropped.

- Designs for a multi-substrate reservoir are introduced.

Reservoirs possess fundamental limitations: due to network coupling, reservoirs cannot model multiple time-scales simultaneously. Furthermore, to solve more complex tasks requires impractically large reservoirs (the *task complexity* problem). In **Chapter 7**, an evolvable hierarchical structure, the *Reservoir-of-Reservoirs* (RoR) architecture, is introduced. Its primary design is to facilitate future multi-substrate reservoir systems. The aim is, through hierarchical organisation, to overcome restrictions in hardware (i.e. adopting multiple smaller systems, rather than creating one large system), model multiple scales, add increased layers of abstraction, and reduce the task complexity problem.

- To demonstrate the concept, a *master* reservoir connecting a network of smaller leaky integrator echo state networks (*sub-reservoirs*) is defined. All parameters within the architecture (except output weights) are open to evolutionary manipulation.

- Two variations of the RoR architecture are investigated: inputs to a single sub-reservoir (RoR) only, and inputs to all sub-reservoirs (RoR-IA).

- Two other multi-reservoir architectures are also evolved for comparison: an ensemble of ESNs and *deep* layered ESNs (DeepESN). For the DeepESN, two variations are defined similar to RoR: inputs-to-one (DeepESN) and inputs-to-all (DeepESN-IA).

- The *steady-state* microbial GA is used to evolve all architectures and parameters. Output training is conducted in the same manner to all previous chapters, reducing the error between reservoir states (from all neurons) and a target signal.

- Two benchmark tasks are applied: the NARMA-10 and NARMA-30 tasks.

- First a baseline is established, evolving *non-hierarchical* ESNs (200-node and 400-node) using the microbial GA. The results show the microbial GA could outperform all other optimisation techniques found in the literature. In some cases, evolved smaller networks significantly outperform larger optimised networks.

- For all hierarchical networks, three network sizes are chosen and fixed (100-node, 200-node and 400-node) split into arbitrary sub-network sizes; 100-node=2×50-nodes, 200-node= 2×100-nodes, 400-node= 4×100-nodes.

- On average for every size and task, the baseline ESN outperforms all hierarchical architectures. However, in some cases, no significant difference in performance is present between the baseline ESN and RoR-IA.

- The general decrease in performance, resulting from adding hierarchical structure, is postulated as an effect of fewer evolvable parameters, since dividing larger networks into smaller sub-networks causes a loss of internal parameters (i.e. *neuron-to-neuron* weights). To compare fairly to a single larger network, an increase in sub-network size is required to remove the deficit in parameters. This is demonstrated with a *compensated* RoR-IA. The results show the new RoR-IA significantly outperforms equivalent, same-sized baseline ESNs.

- Despite performing less well than the non-hierarchical ESN, all evolved hierarchical architectures still outperform other hierarchical and non-hierarchical architectures in the literature.

- In the final experiment, the hypothesis that hierarchy can improve task generalisation is tested. Two further benchmarks (*Santa Fe laser time-series prediction* and the *Hénon map* map task) are evaluated on the best evolved architectures from both the NARMA-10 and NARMA-30 experiments. The results showed RoR-IA generalises better to each new task, despite size and original task. The result suggests something within the RoR structure improves generalisation, which also appears to increase with network size.

In **Chapter 8**, the new SQuARC framework (built on the RCiM framework) provides a method to measure the *quality* of any substrate for reservoir computing. The hypothesis is: substrates that can exhibit more distinct behaviours have a greater dynamical freedom, which implies they can realise more distinct reservoirs, thus have a greater quality for reservoir computing. The framework fills a significant gap in both the RC and EiM field, providing a method to compare different reservoir systems (both *in silico* and *in materio*), and to understand what computational properties are required to compute specific tasks and exhibit a generalised computing ability.

- The SQuARC framework is defined by five layers (and two phases); 1) *Definition*, 2) *Exploration and mapping*, 3) *Evaluation*, 4) *Learning*, and 5) *Application*. The first phase requires the assessment of a *reference* substrate, and the second phase assesses the desired *test* substrate.
- The first phase of the framework is applied to echo state networks, forming the reference substrate.
- To measure quality, an abstract task/substrate-independent space is defined, called the *behaviour space*. It represents reservoir behaviours (measured as approximations of dynamical properties) of different substrate configurations. The applied task-independent measures of dynamical properties are given in section 8.3.
- The behaviour space to be explored is defined by three axes; *kernel rank* (KR), *generalisation rank* (GR), and *Memory capacity* (MC).
- To measure how many distinct behaviours are found (the total space covered) the space is divided into *behaviour voxels*.

- Novelty Search, the algorithm used to map and explore the behaviour space, is described in section 8.4.3 and its implementation in section 8.4.2.

- Results of *random* search versus novelty search show the latter has a greater ability to explore the behaviour space in the same time. This is demonstrated using four ESN sizes; 25, 50, 100 and 200-nodes.

- The effect network size has on the rate of exploration (*novelty rate*) is investigated. It is found that for larger networks novelty search explores the space faster (as the rate increases), whereas the rate for random search decreases steadily with time. The difference observed between the larger and smaller networks' novelty rate is postulated as being the result smaller networks converging faster. With fewer new distinct behaviours attainable as time passes – as novelty also becomes more difficult to obtain – the rate of smaller networks decreases faster.

- In section 8.5.4, novelty search's ability to (re)discover general parameter "rules" is discussed. As an example, common parameter rules associated with ESNs are highlighted and a potential new rule is discovered; the inverse relationship between connectivity of W and W_{in} as ESN size increases.

- To test the accuracy of the property measures, and determine whether they *faithfully* represent computation within the substrate, the *property-task* relationship is modelled. To model the relationship, every behaviour discovered from every ESN size is evaluated on a series of tasks. The four tasks used are: NARMA-10, NARMA-30, Santa Fe laser, and the non-linear channel equalisation task. Data from each ESN size is then used to train individual feed-forward neural networks (FFNNs) and predict performance across different network sizes.

- Low prediction errors found with different tasks and ESN sizes show the property measures represent a faithful approximation. For some tasks, however, prediction is harder than others (e.g. non-linear channel equalisation), demonstrating the non-trivial problem of modelling property-task relationships. The results also show that when the ESN size to be predicted is greater than the training ESN size, prediction error can be high.

- In the second phase, the new 64-electrode hardware platform is used. The substrate under-test is the newly made carbon nanotube/polymer substrate (1% SW-CNT/polymer mixture). Results of the quality assessment show the physical substrate exhibits limited behaviour compared to small (roughly equivalent) ESNs.
- The poor quality of the physical substrate is investigated further. Using the *novelty rate*, it is determined that the dynamical boundaries/limits of the substrate have been reached.
- In the final stage of the framework, the prediction transfer (i.e. ability to predict performance of one substrate based on another) from reference substrate to physical substrate is measured. The hypothesis is: the prediction transfer error (Δ) between substrates can quantify the substrate-independence of the framework. The results show on average a small Δ is present when predicting the physical substrate's task performance, using FFNNs trained with ESNs of all sizes. The non-linear channel equalisation task proved more difficult to predict than other tasks. The best ESN size to use as a reference is determined to be the larger 200-node ESN, suggesting a simple and intuitive rule: the quality of the reference substrate determines the accuracy the FFNN's prediction, with greater quality resulting in greater accuracy.

9.2 Future Work

Although all the *in materio* work presented uses carbon nanotube composites, the substrate acts only as a *proxy* to test the reservoir computing *in materio* concept. Each framework proposed is designed to generalise to any physical (and in some cases virtual) substrate. This leads to a vast amount of open research as a result of the theses, such as testing new substrates and adjusting/updating frameworks accordingly, and using the frameworks to help design better substrates.

Future work associated with more specific parts of the thesis are as follows:

9.2.1 The RoR Architecture

The RoR architecture in Chapter 7 shows promising results, however, many aspects of the architecture can be improved, and many questions are still left unanswered:

- A more in-depth investigation is required to understand the effect *sub-reservoir* size has on RoR architecture performance and transfer learning. In experiments, the chosen sub-reservoir sizes were arbitrary. What was discovered is that master reservoirs with more sub-reservoirs tend to show greater task generalisation. However, a general "rule" is still missing. This leaves the question; what is the *threshold*, or ideal *ratio*, at which the number of sub-reservoirs degrades performance and task generalisation?

- Little is still known about the architecture's homogeneity/heterogeneity. More tasks and analysis are required to determine the architecture's ability to model multiple scales.

- Before any *in materio* experiments, multiple *in silico* sub-reservoirs should be combined and tested. For example, instead of using only ESNs, combine other random recurrent networks (e.g. long-short term memory (LSTMs) [80], gated recurrent units (GRUs) [32], liquid state machines (LSMs)), non-recurrent networks (random feed-forward networks/extreme learning machines (ELM) [83, 84]), and other dynamical systems (e.g. random boolean (RBN), gene regulatory networks (GRN) [95], cellular automata (CA) [202]).

- Redundancy could be a key advantage to the RoR architecture. This conjecture, however, still needs testing. For example, what effect does removing, or isolating, a sub-reservoir have? Can connections in the master reservoir be removed after training? Would regularisation and pruning improve the evolutionary search and reduce overall reservoir size? Examples include using "dropout" [173] (dropping out units) to remove redundant nodes, pruning of the readout [52], or starting evolution with minimal complexity [174].

9.2.2 Heterotic Computing

Heterotic computing [96, 182] is defined as a combination of two or more computational systems such that they provide an advantage over either substrate used separately. The hypothesis is that different computational models can be combined to form a more powerful non-classical computing system. The inclusion of different models therefore leaves different computational devices to do what they do naturally, and best.

The original rationale for the RoR architecture was for multi-substrate reservoir computing. In some ways, it mirrors the same principles outlined in heterotic computing. However, all substrates umbrella under the reservoir computing framework, i.e. are represented as *reservoirs*. In future work, a merger in ideas and techniques between multi-substrate reservoir computing and heterotic computing could offer new unique computing systems. In Chapter 8, ideas from Abstraction/Representation Theory [82], used to formulate heterotic computing, have already been put to good use.

9.2.3 Reservoir Computing *in materio* Framework

The RCiM framework is proven in Chapter 5 to have benefits over EiM. However, there are many ways to improve and future-proof the RCiM framework:

- Design new input encodings for tasks that require multiple inputs. So far, only a few experiments have been conducted with multi-input tasks, and as such, only a rudimentary input strategy is used. In the future, tasks with greater input-output complexity will be desired. This may include, high-input classification tasks such as image and speech, natural language processing, sequence generation, dimensionality reduction, cryptography, etc.

- A potentially significant problem for *in materio* systems in the future is an ability to retain information intrinsically for long-periods of time. However, methods of storing information in analogue mediums are bountiful. The potential danger is adopting more classical techniques to solve the long-term memory problem, thus bringing the bottlenecks associated with it. How best to achieve this is still largely unknown, and perhaps even unnecessary; a hybridisation with digital computers could remove the problem. An alternative solution for longer-memory could come from adopting hybrid principles presented in Neural Turing Machines (NTMs) [66] and the Differentiable Neural Computer (DCN) [67], where neural networks are coupled to *external* memory resources which they can interact with by attentional processes.

- To solve the more immediate short-term memory problem, experienced by *weak* memory substrates, several avenues are possible: adding various levels of feedback, which has been here largely unexplored, could promote oscillatory dynamics and thus improve memory. This could include, delayed feedback through analogue components, or virtually in the digital domain. Another avenue may be to create temporary artificial memory cells, storing and relaying past information back into the substrate (physically or virtually) using state update equations similar to long-short term memory networks (LSTMs) [80] and gated recurrent units (GRUs) [32].
- Greater hardware expansion, and larger *in materio* reservoir sizes, are still possible. As technology improves, the current system can be expanded. However, other means of measurement (and configuration) may be desirable. The flexibility of both the evolution *in materio* and reservoir computing frameworks allow for a vast variety of methods (and combinations) to observe states and evolve parameters. Lessons learnt about substrate isolation from Chapter 6 may prove invaluable to any future expansion.
- An all analogue implementation is still desired, where the readout/output weights are realised in hardware. In such an implementation, making the readout evolvable would also remove the supervised training process. However, this could also lead to greater genotype-phenotype complexity. Achieving an all-analogue implementation could make the whole RCiM concept more viable to real-world applications.

9.2.4 The SQuARC framework

The SQuARC framework's modular design makes it easy to enhance and update. Examples of future experiments with the framework include:

- Enforce minimal criteria to the search for novelty. In [109], solutions not satisfying the minimal criteria are given a low fitness/novelty, adding a selection pressure towards more functional solutions. It is a method to prune the “space of viable behaviours”, and is found to be more efficient than searching for novelty alone. To achieve this in the framework, one might require some dynamical property exist, or a particular physical constraint to be adhered to.
- Test the SQuARC framework on different physical and virtual substrates. Once new substrates are collected and compared, a repository could be created. This would feature reference substrates and task prediction systems for general use by the wider community.
- Apply the *RoR* architecture to the SQuARC framework. The framework may require adjusting to compensate for multi-reservoir architectures, as multiple sets of a dynamics will need to be captured.
- Add more reservoir *metrics* and map more complex tasks to the behaviour space. This should improve the accuracy of the framework and our understanding of required computational properties needed to compute both simple and complex tasks.
- Utilise the behaviour space and genetic representation to better understand the physical interactions being exploited. This might consist of modelling the relationships between behaviour space and inputs/outputs/control signals, or, visualising the connectivity of the underlying structure using the strength of signals between electrodes.

9.2.5 Framework-in-the-loop Hardware Design

Due to the relatively automated process of assessing the quality of substrates via the SQuARC framework, new avenues could open for substrate design.

In the future, the design and fabrication of the substrate may also be under evolutionary control, resulting in computers that can design other computers from the bottom-up. This is made possible by the quality measure providing a method to compare substrates, therefore quality could be optimised, or explored autonomously.

One way to implement this could be through additive design/manufacturing processes, such as the 3D printing of substrates, with the potential to evolve both the substrate and the interface design (e.g. electrodes). Other implementations could involve the evolution of chemical or biological substrates, with the potential of rapidly improving the computing capability of unconventional computing systems.

Appendix A

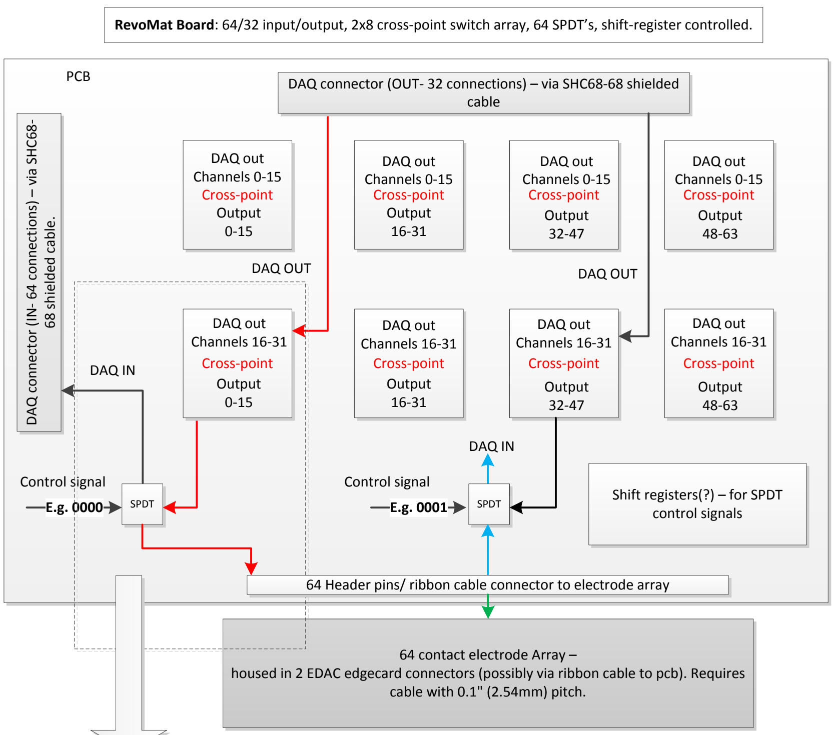
Hardware Platform Designs

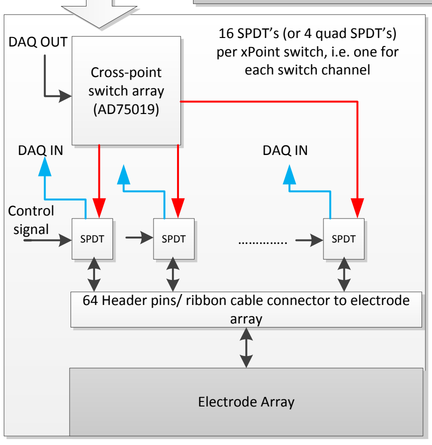
In this appendix are ideas and designs for different components related to the new hardware platform.

Early ideas of how to connect and route the electrodes to the DAQ cards are shown in A.1. The final designs for the second iteration of the routing board are given in A.2. In A.3, different 64 electrode array designs and masks are shown with various electrode spacings and contact sizes. These designs were made for etch-back photo-lithography on glass slides, however they were never used. A.4 gives I-V curve plots of various carbon nanotube composites investigated within this work. These also include the effect of a bias on the current vs. voltage relationship.

In the last section A.5, the initial ideas for a multi-substrate housing are given, along with details of the LED arrays used in Chapter 6.

A.1 Routing Board Ideas


RevoMat Board: 64/32 input/output, 2x8 cross-point switch array, 64 SPDT's, shift-register controlled.
PCB
DAQ connector (OUT- 32 connections) – via SHC68-68 shielded cable
DAQ connector (IN- 64 connections) – via SHC68-68 shielded cable.
DAQ out Channels 0-15 Cross-point Output 0-15
DAQ out Channels 0-15 Cross-point Output 16-31
DAQ out Channels 0-15 Cross-point Output 32-47
DAQ out Channels 0-15 Cross-point Output 48-63
DAQ OUT
DAQ OUT
DAQ out Channels 16-31 Cross-point Output 0-15
DAQ out Channels 16-31 Cross-point Output 16-31
DAQ out Channels 16-31 Cross-point Output 32-47
DAQ out Channels 16-31 Cross-point Output 48-63
DAQ IN
DAQ IN
Control signal
E.g. 0000
SPDT
Control signal
E.g. 0001
SPDT
Shift registers(?) – for SPDT control signals
64 Header pins/ ribbon cable connector to electrode array
64 contact electrode Array – housed in 2 EDAC edgecard connectors (possibly via ribbon cable to pcb). Requires cable with 0.1" (2.54mm) pitch.


DAQ OUT
Cross-point switch array (AD75019)
16 SPDT's (or 4 quad SPDT's) per xPoint switch, i.e. one for each switch channel
DAQ IN
DAQ IN
Control signal
SPDT
SPDT
....
SPDT
64 Header pins/ ribbon cable connector to electrode array
Electrode Array

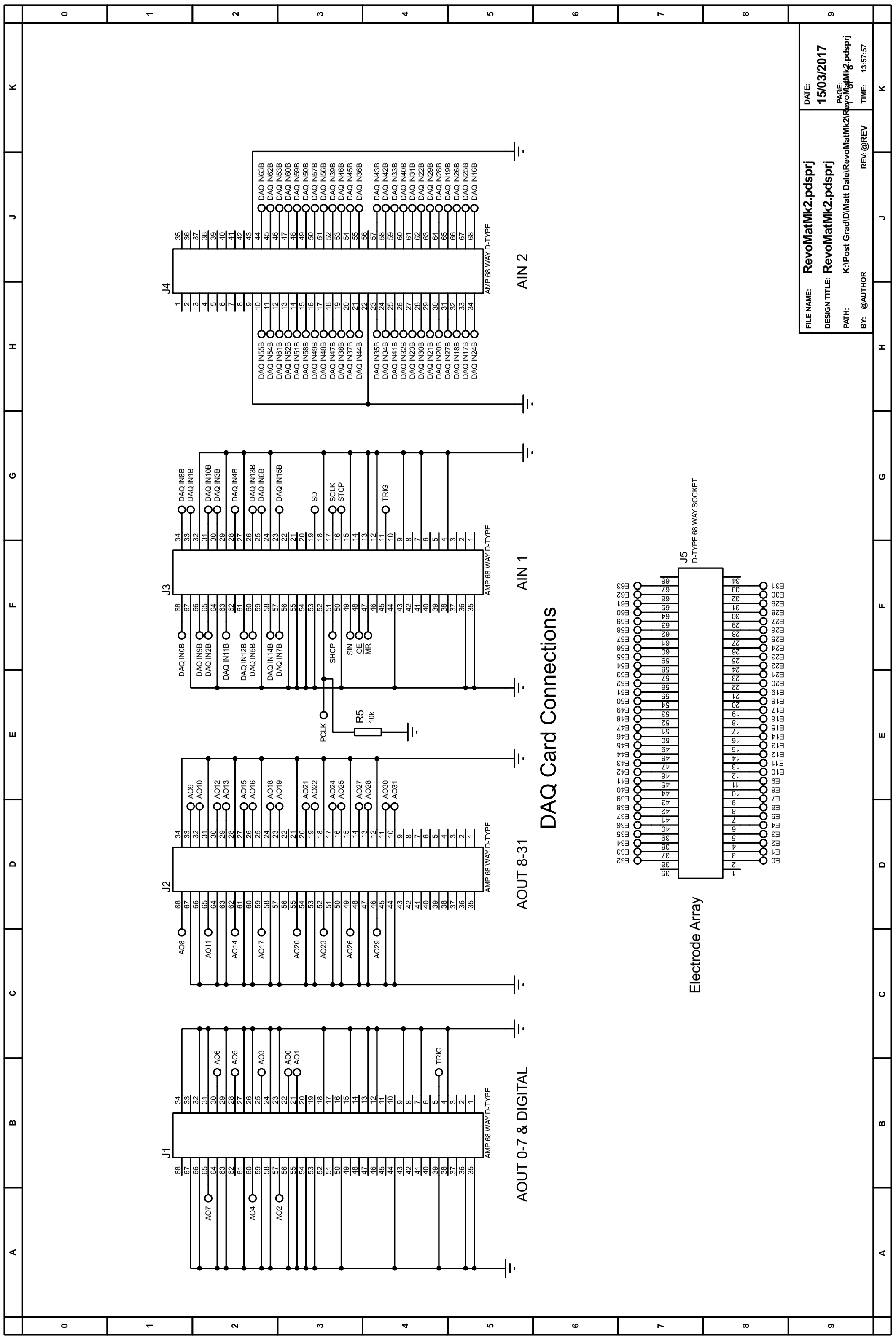
Total DAQ channels needed:

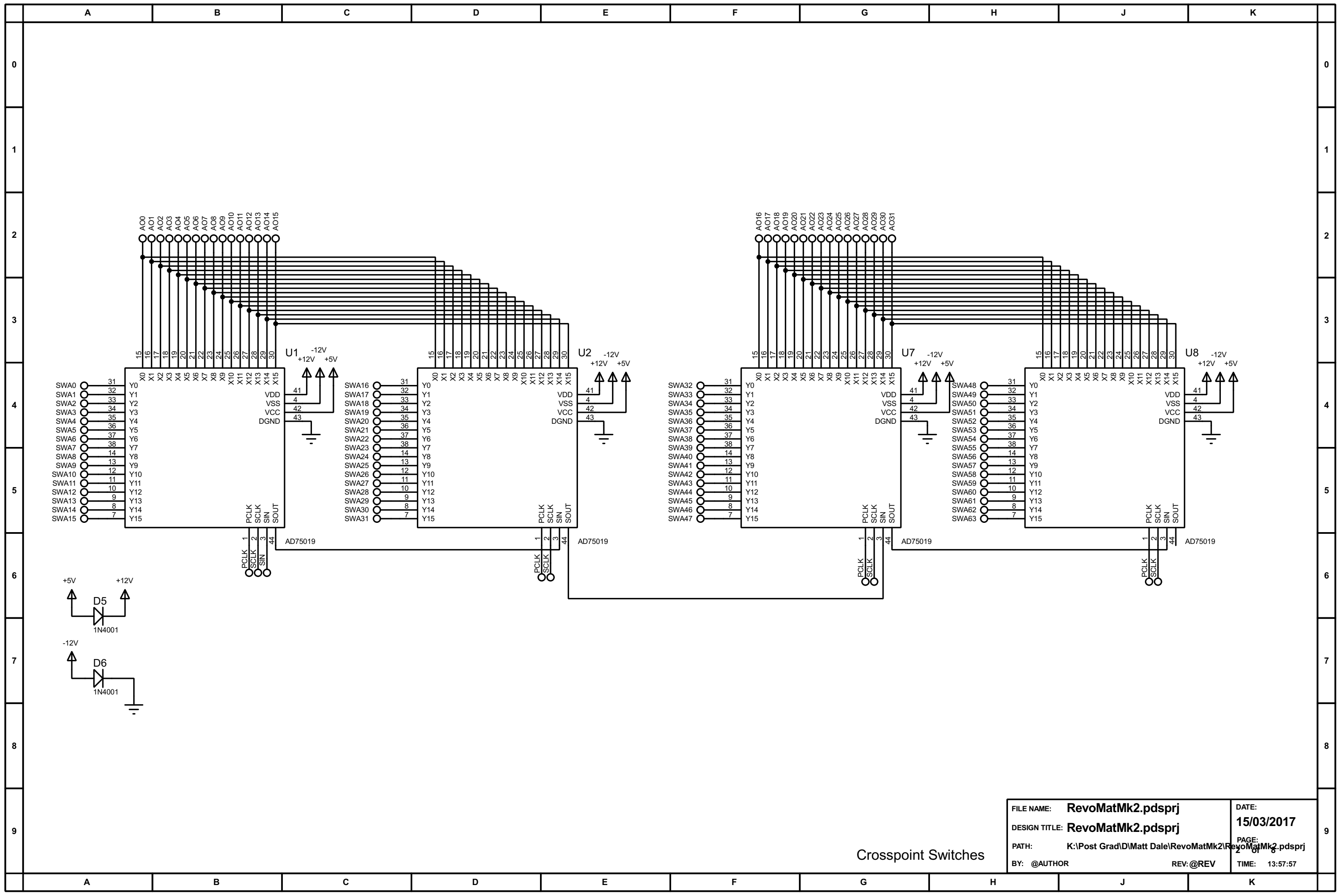
- 32 outputs
- 64 inputs
- 1 digital trigger (connecting both cards)
- 5V (available, if necessary)
- DGND
- SCLK
- PCLK
- SIN
- Shift register signals (depends on choice): "Data In", "CLK", "Latch", "Clear", "Data out", etc. (NI PCI-6225 has 24 digital I/O)

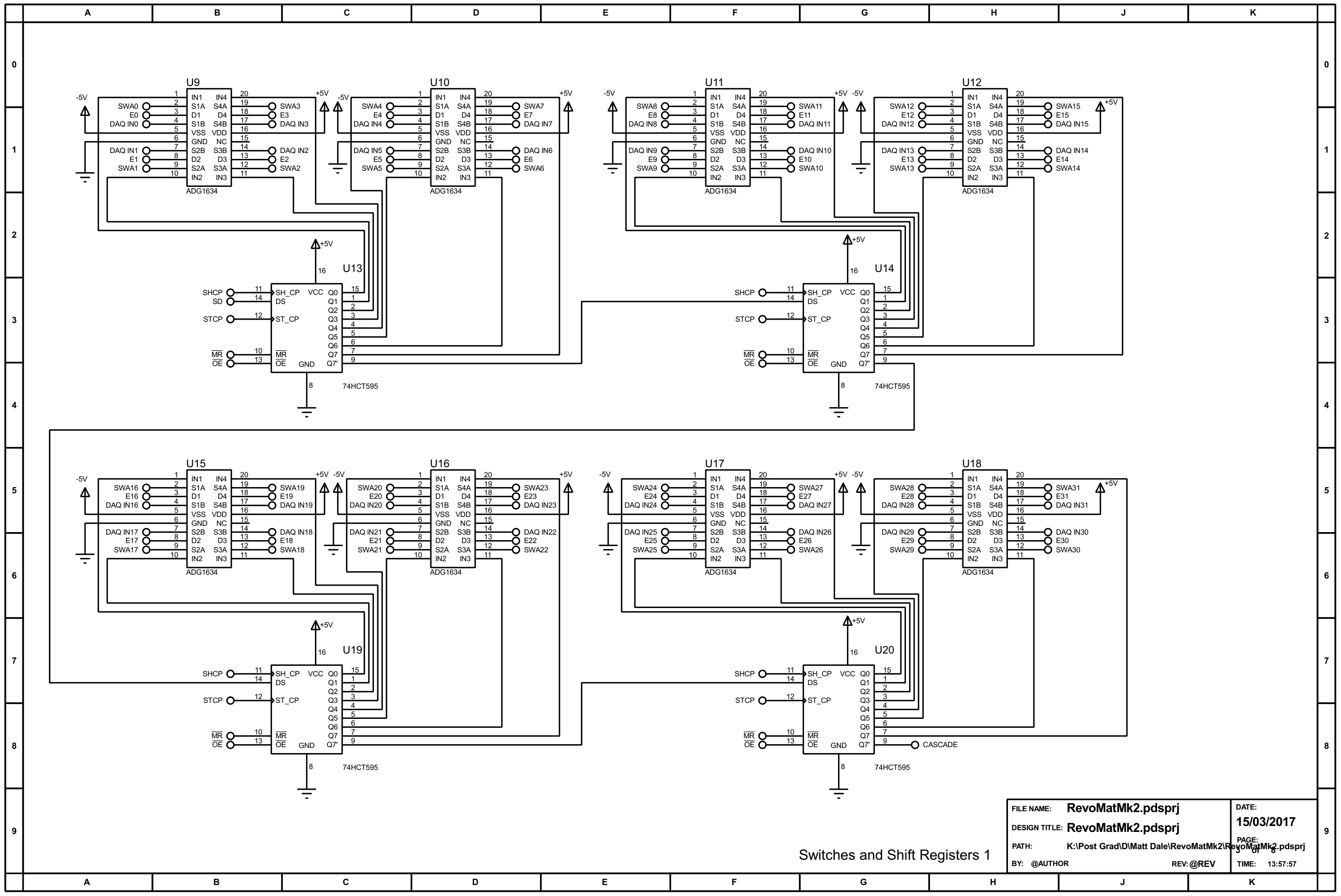
Notes:

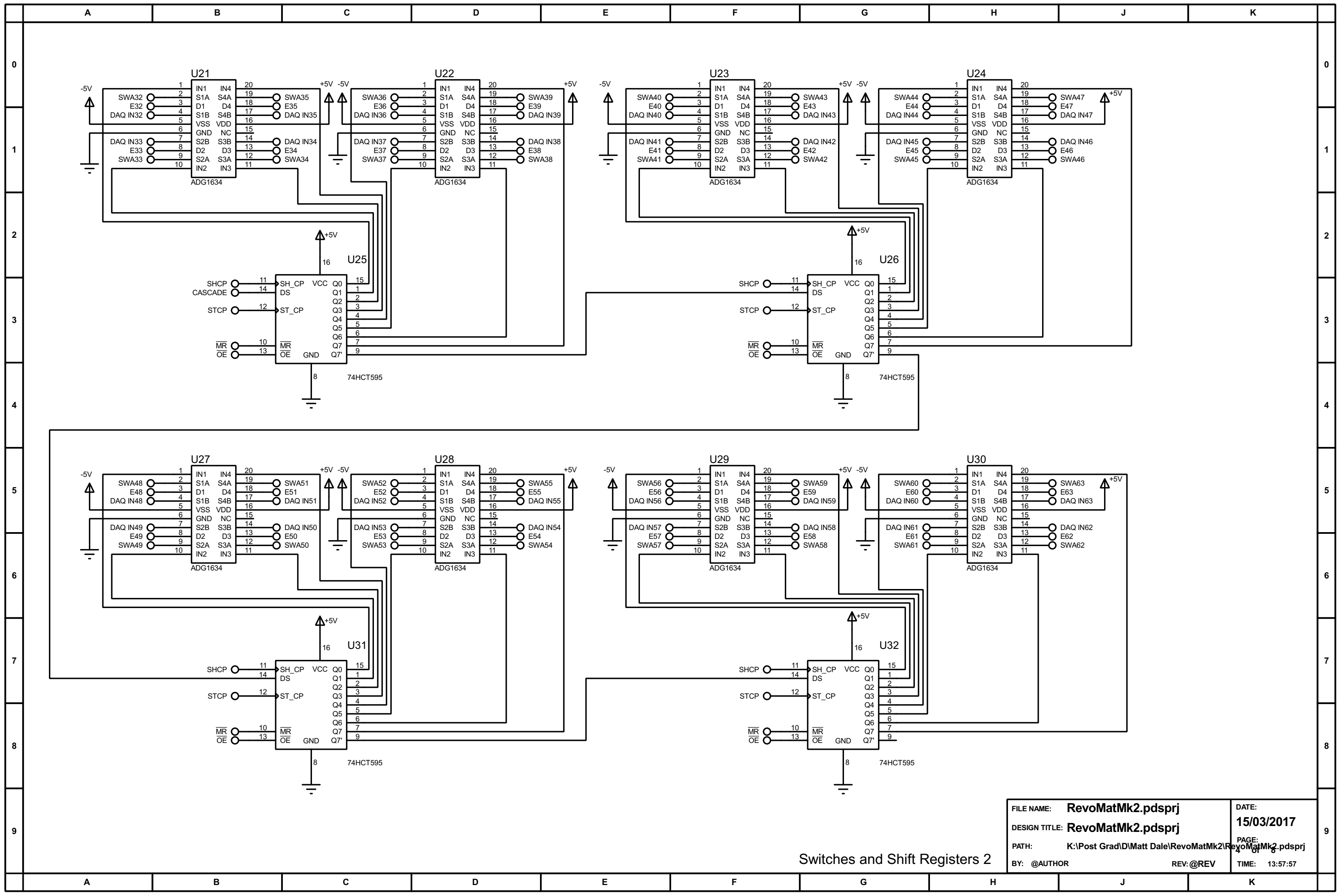
- In total 64 analogue switches are required, e.g. (2:1) SPDT's. This could however be easily integrated into (2:1) quad SPDT's (e.g. an ADG1634 or ADG1234/ADG1334). In a setup using quads only 16 ADG1634's would be required.
- A two-by-four array (8 total) of Cross-point switch array's (AD75019) is needed – currently possess 8 AD75019JPZ's. This will take in analogue voltages supplied by an National Instruments PCI-6723 DAQ (purchased). Switches can be daisy chained via SOUT.
- The switches connected to each electrode (e.g. ADG1634's) should connect to the AD75019 and directly to the NI PCI-6225 (purchased) connections. This could be through the NI SHC68-68 shielded cable/connector.
- An ADG1634 is suggested as it has a dual power supply +/- 5V, and therefore an analogue signal range between +/- 5V. The inputs from each AD75019 channel should be no bigger than +/- 5V (defined by the DAQ), and the voltage drop across the material at the readout should be significantly less. Although, the ADG1334 has a higher dual power range between +/- 15V – which is closer to the AD75019 (and cheaper!), but has a higher internal on resistance (130Ohm compared to 4.5Ohm). Suggestions would be appreciated.
- It has been suggested the power supplies to the AD75019's (+12V,-12V,+5V) require 10uF and 0.1uF capacitors.
- Ideally power to the main board could be something like a single +12V input, with inverters and 5V regulator (although the DAQ does provide a 5V line) added to power the AD75019's/SPDT's.
- PCLK to the AD75019's probably requires a 10KOhm pull-down.
- I currently have two EDAC card connectors (Part No. 345-064-520-202) to interface with the electrode array. These are dual 32-pin card connectors (one side won't be used). I think the best solution (but not ideal) solution is to connect the EDAC connector to the PCB via something like ribbon cable because, at the moment, the EDAC's connect to either side of a glass slide. Therefore, soldering the connectors to the board would make inserting a slide, difficult.
- *I've done little research on this, so far*: To control the SPDT's a 64-bit (or 32-bit x2, 16-bit x4, 8-bit x 8) shift register(s) could be used.

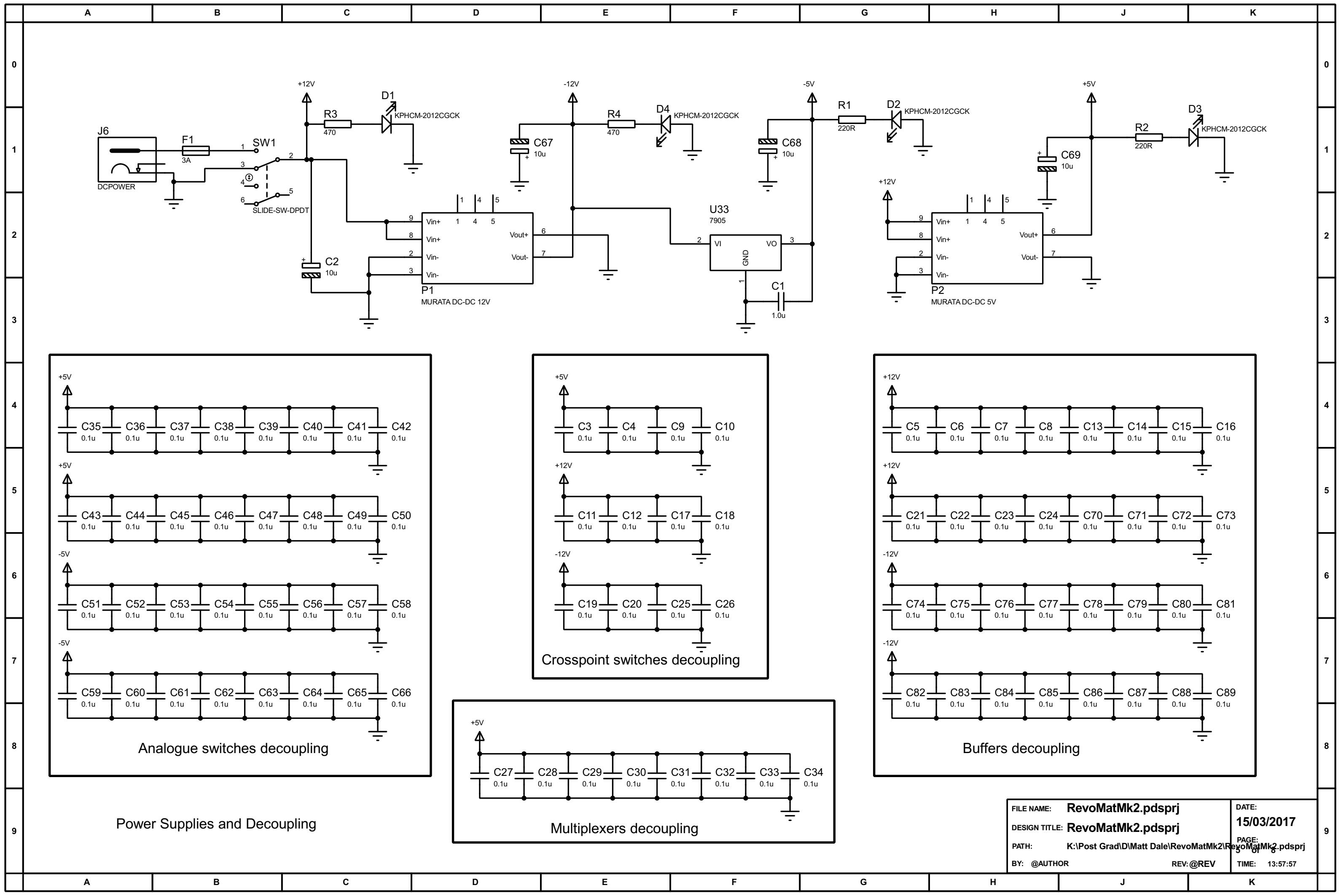
A.2 Routing Board Designs

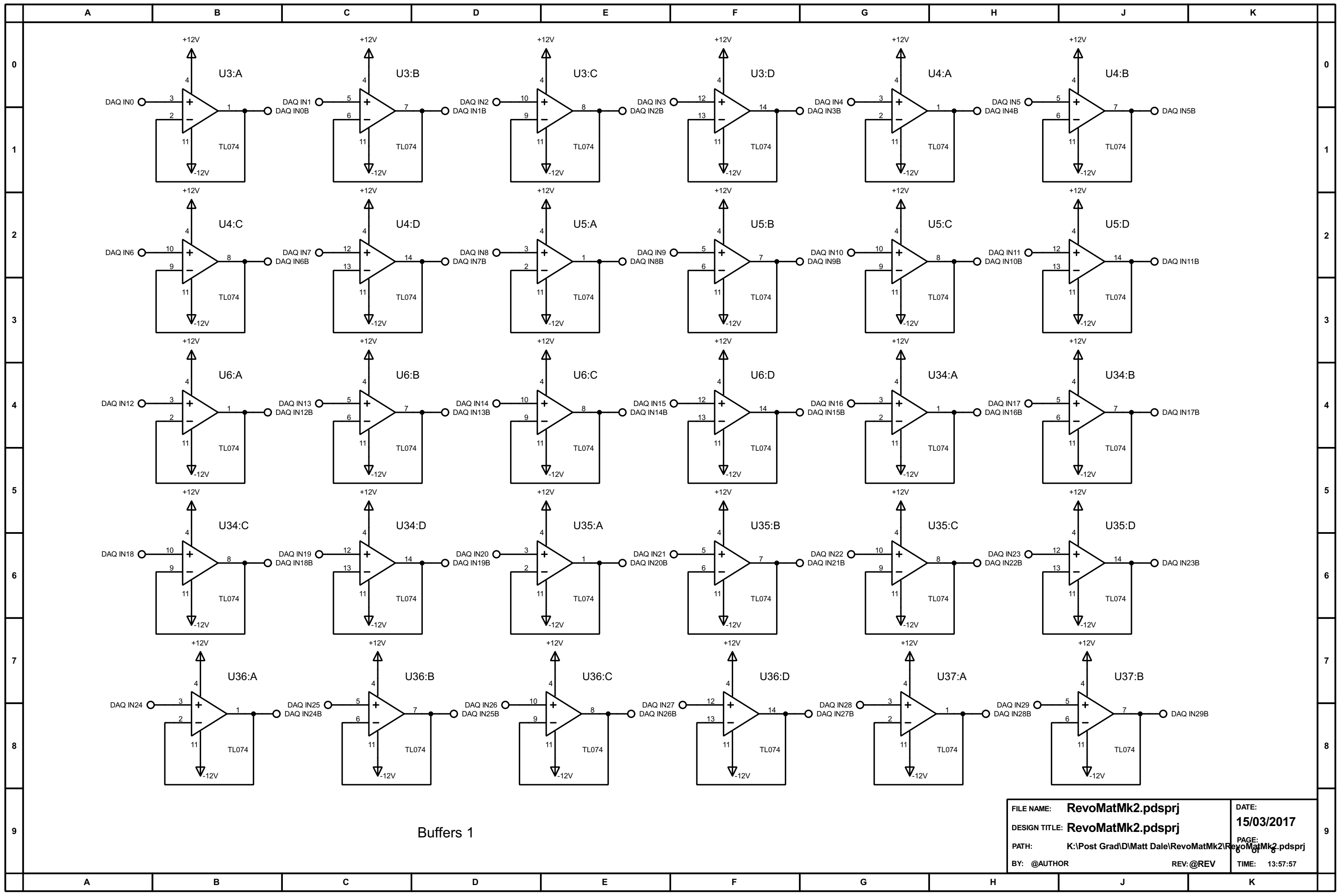


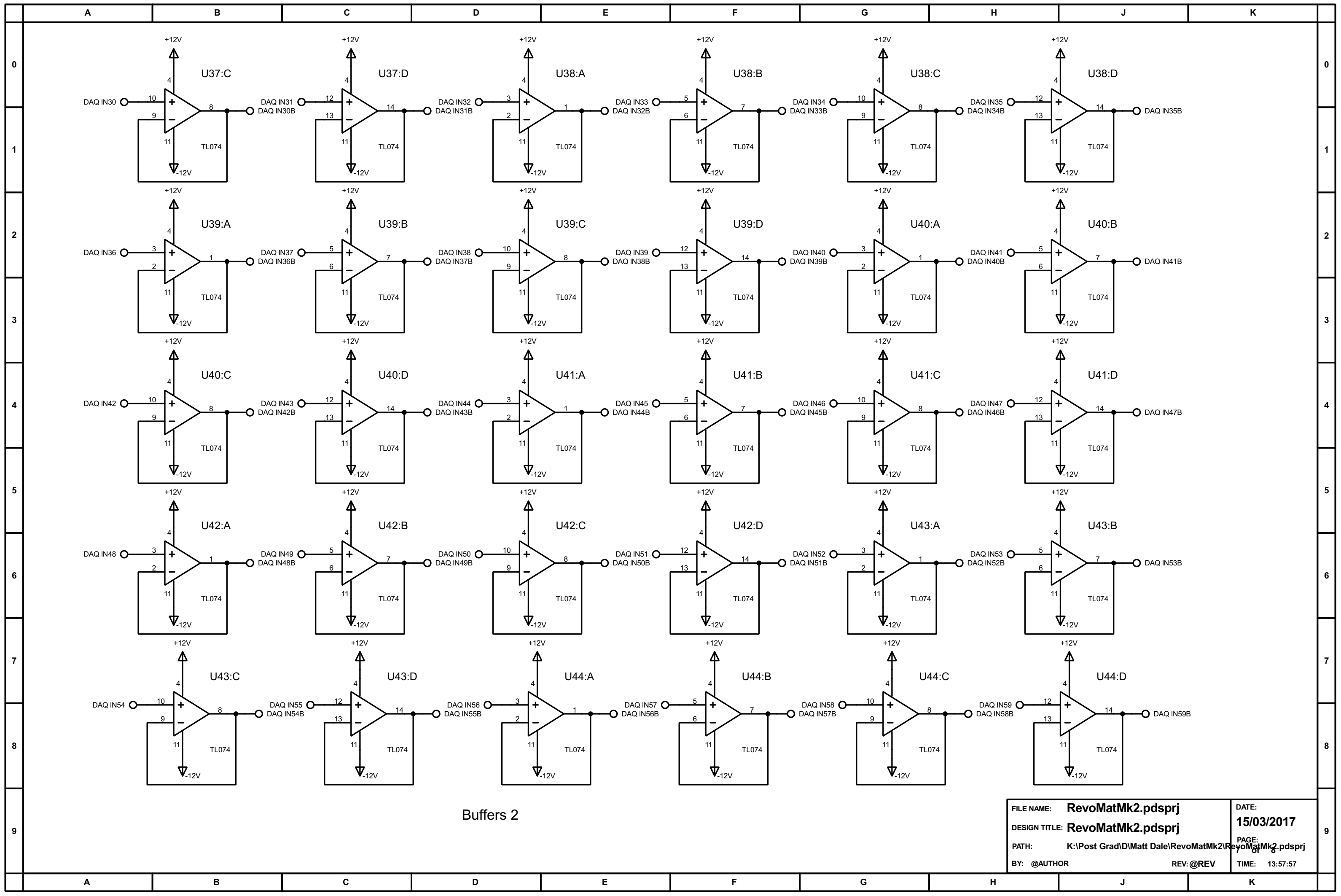

U1
U2
U7
U8
AD75019
AO0–AO15
AO16–AO31
SWA0–SWA15
SWA16–SWA31
SWA32–SWA47
SWA48–SWA63
X0–X15
Y0–Y15
VDD
VSS
VCC
DGND
PCLK
SCLK
SIN
SOUT
+12V
-12V
+5V
D5
D6
1N4001
Crosspoint Switches
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 2 of 6
TIME: 13:57:57

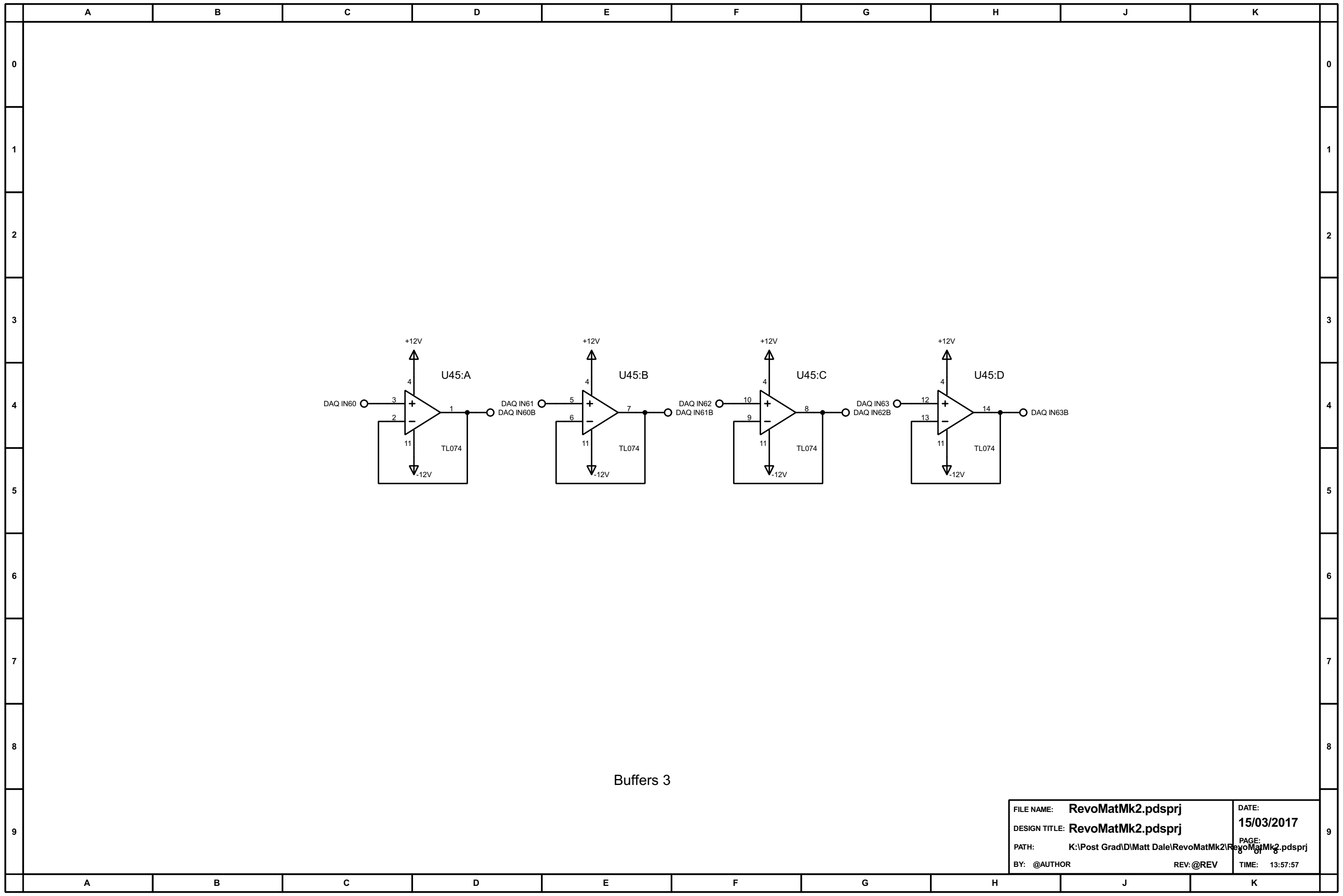

U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
ADG1634
74HCT595
SHCP
SD
STCP
MR
OE
CASCADE
+5V
-5V
Switches and Shift Registers 1
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 3 of 8
TIME: 13:57:57


U21
U22
U23
U24
U25
U26
U27
U28
U29
U30
U31
U32
ADG1634
74HCT595
IN1
IN2
IN3
IN4
S1A
S1B
S2A
S2B
S3A
S3B
S4A
S4B
D1
D2
D3
D4
VSS
VDD
GND
NC
SH_CP
ST_CP
DS
VCC
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
Q7'
MR
OE
+5V
-5V
SHCP
STCP
CASCADE
SWA32
E32
DAQ IN32
DAQ IN33
E33
SWA33
SWA35
E35
DAQ IN35
DAQ IN34
E34
SWA34
SWA36
E36
DAQ IN36
DAQ IN37
E37
SWA37
SWA39
E39
DAQ IN39
DAQ IN38
E38
SWA38
SWA40
E40
DAQ IN40
DAQ IN41
E41
SWA41
SWA43
E43
DAQ IN43
DAQ IN42
E42
SWA42
SWA44
E44
DAQ IN44
DAQ IN45
E45
SWA45
SWA47
E47
DAQ IN47
DAQ IN46
E46
SWA46
SWA48
E48
DAQ IN48
DAQ IN49
E49
SWA49
SWA51
E51
DAQ IN51
DAQ IN50
E50
SWA50
SWA52
E52
DAQ IN52
DAQ IN53
E53
SWA53
SWA55
E55
DAQ IN55
DAQ IN54
E54
SWA54
SWA56
E56
DAQ IN56
DAQ IN57
E57
SWA57
SWA59
E59
DAQ IN59
DAQ IN58
E58
SWA58
SWA60
E60
DAQ IN60
DAQ IN61
E61
SWA61
SWA63
E63
DAQ IN63
DAQ IN62
E62
SWA62
Switches and Shift Registers 2
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 4 of 6
TIME: 13:57:57

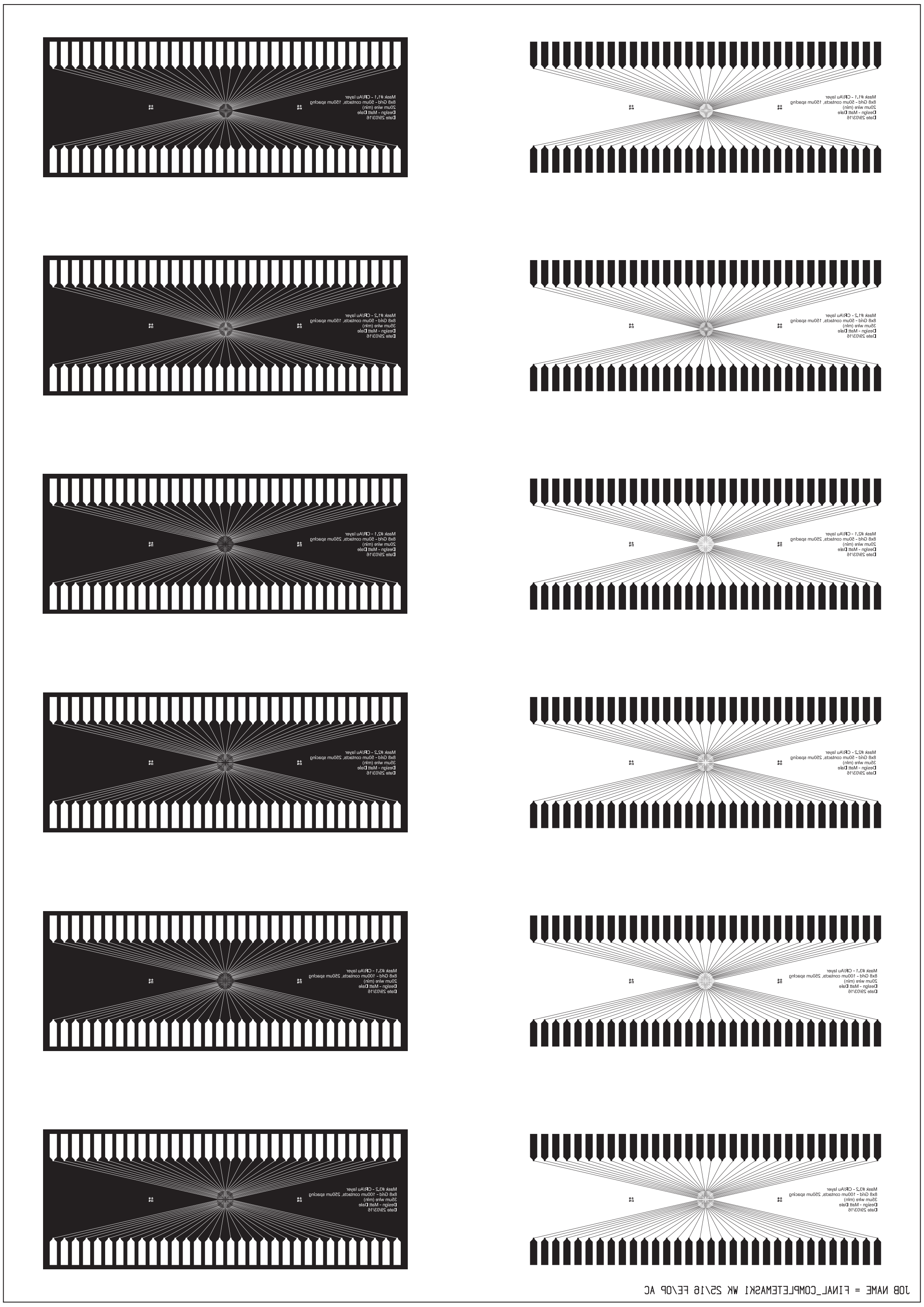

J6
DCPOWER
F1
3A
SW1
SLIDE-SW-DPDT
+12V
R3
470
D1
KPHCM-2012CGCK
C2
10u
P1
MURATA DC-DC 12V
Vin+
Vin-
Vout+
Vout-
-12V
C67
10u
R4
470
D4
KPHCM-2012CGCK
U33
7905
VI
VO
GND
C1
1.0u
-5V
C68
10u
R1
220R
D2
KPHCM-2012CGCK
+5V
C69
10u
R2
220R
D3
KPHCM-2012CGCK
P2
MURATA DC-DC 5V
Analogue switches decoupling
C35 C36 C37 C38 C39 C40 C41 C42
C43 C44 C45 C46 C47 C48 C49 C50
C51 C52 C53 C54 C55 C56 C57 C58
C59 C60 C61 C62 C63 C64 C65 C66
0.1u
Crosspoint switches decoupling
C3 C4 C9 C10
C11 C12 C17 C18
C19 C20 C25 C26
Multiplexers decoupling
C27 C28 C29 C30 C31 C32 C33 C34
Buffers decoupling
C5 C6 C7 C8 C13 C14 C15 C16
C21 C22 C23 C24 C70 C71 C72 C73
C74 C75 C76 C77 C78 C79 C80 C81
C82 C83 C84 C85 C86 C87 C88 C89
Power Supplies and Decoupling
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 5 of 6
TIME: 13:57:57

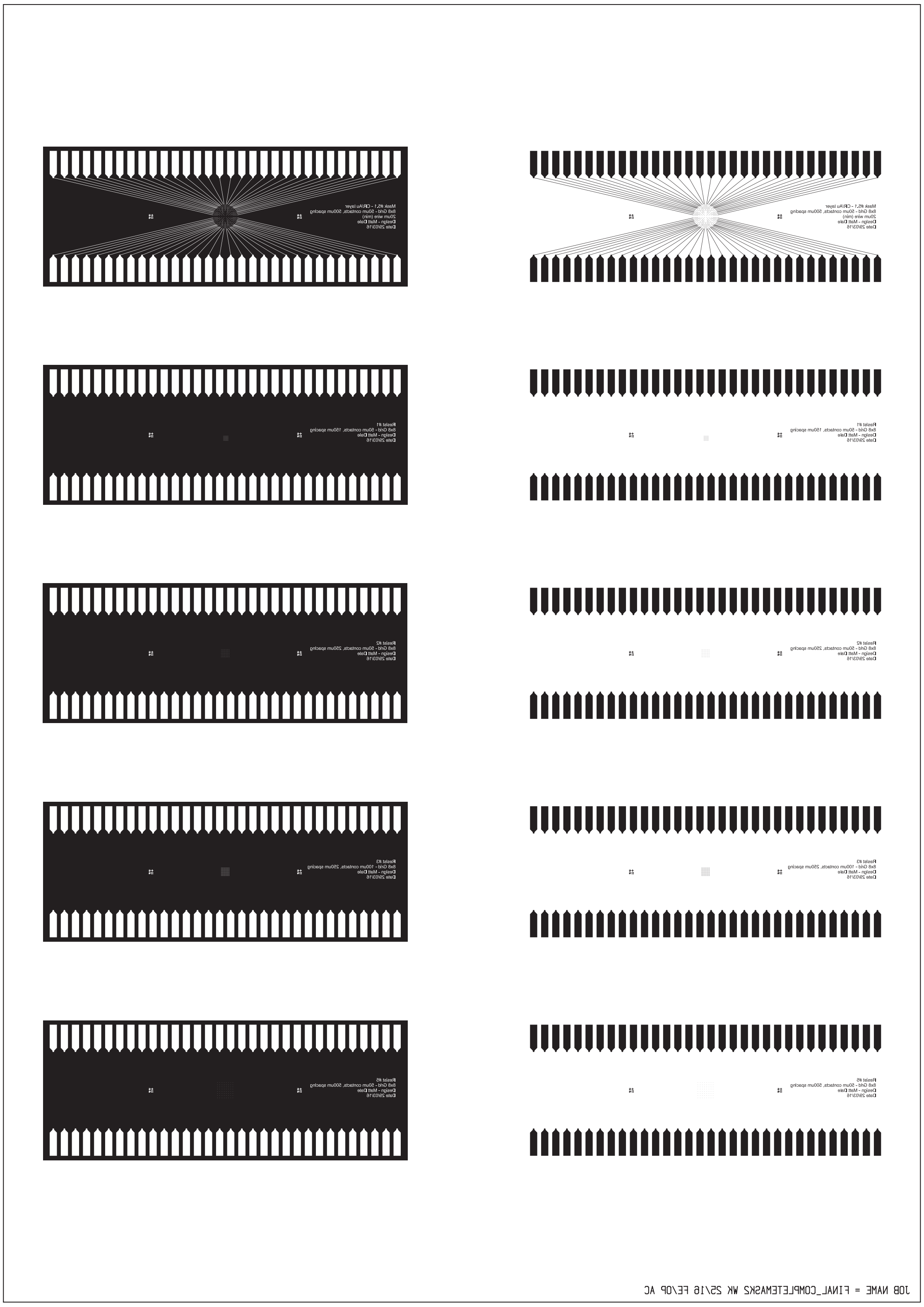

U3:A
U3:B
U3:C
U3:D
U4:A
U4:B
U4:C
U4:D
U5:A
U5:B
U5:C
U5:D
U6:A
U6:B
U6:C
U6:D
U34:A
U34:B
U34:C
U34:D
U35:A
U35:B
U35:C
U35:D
U36:A
U36:B
U36:C
U36:D
U37:A
U37:B
TL074
+12V
-12V
DAQ IN0
DAQ IN0B
DAQ IN1
DAQ IN1B
DAQ IN2
DAQ IN2B
DAQ IN3
DAQ IN3B
DAQ IN4
DAQ IN4B
DAQ IN5
DAQ IN5B
DAQ IN6
DAQ IN6B
DAQ IN7
DAQ IN7B
DAQ IN8
DAQ IN8B
DAQ IN9
DAQ IN9B
DAQ IN10
DAQ IN10B
DAQ IN11
DAQ IN11B
DAQ IN12
DAQ IN12B
DAQ IN13
DAQ IN13B
DAQ IN14
DAQ IN14B
DAQ IN15
DAQ IN15B
DAQ IN16
DAQ IN16B
DAQ IN17
DAQ IN17B
DAQ IN18
DAQ IN18B
DAQ IN19
DAQ IN19B
DAQ IN20
DAQ IN20B
DAQ IN21
DAQ IN21B
DAQ IN22
DAQ IN22B
DAQ IN23
DAQ IN23B
DAQ IN24
DAQ IN24B
DAQ IN25
DAQ IN25B
DAQ IN26
DAQ IN26B
DAQ IN27
DAQ IN27B
DAQ IN28
DAQ IN28B
DAQ IN29
DAQ IN29B
Buffers 1
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 6 of 8
TIME: 13:57:57

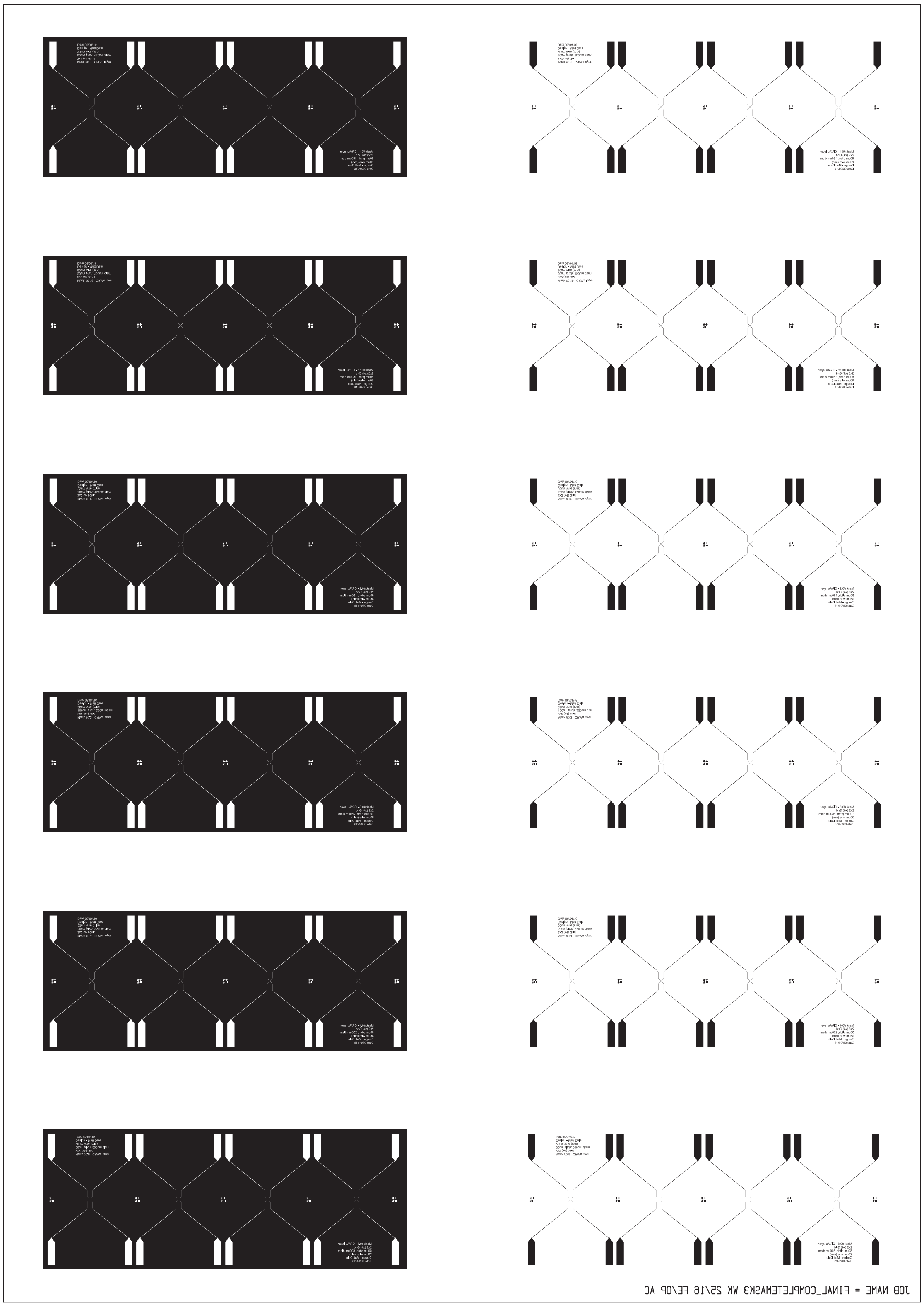

U37:C
U37:D
U38:A
U38:B
U38:C
U38:D
U39:A
U39:B
U39:C
U39:D
U40:A
U40:B
U40:C
U40:D
U41:A
U41:B
U41:C
U41:D
U42:A
U42:B
U42:C
U42:D
U43:A
U43:B
U43:C
U43:D
U44:A
U44:B
U44:C
U44:D
TL074
+12V
-12V
DAQ IN30 … DAQ IN59
DAQ IN30B … DAQ IN59B
Buffers 2
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 7 of 8
TIME: 13:57:57

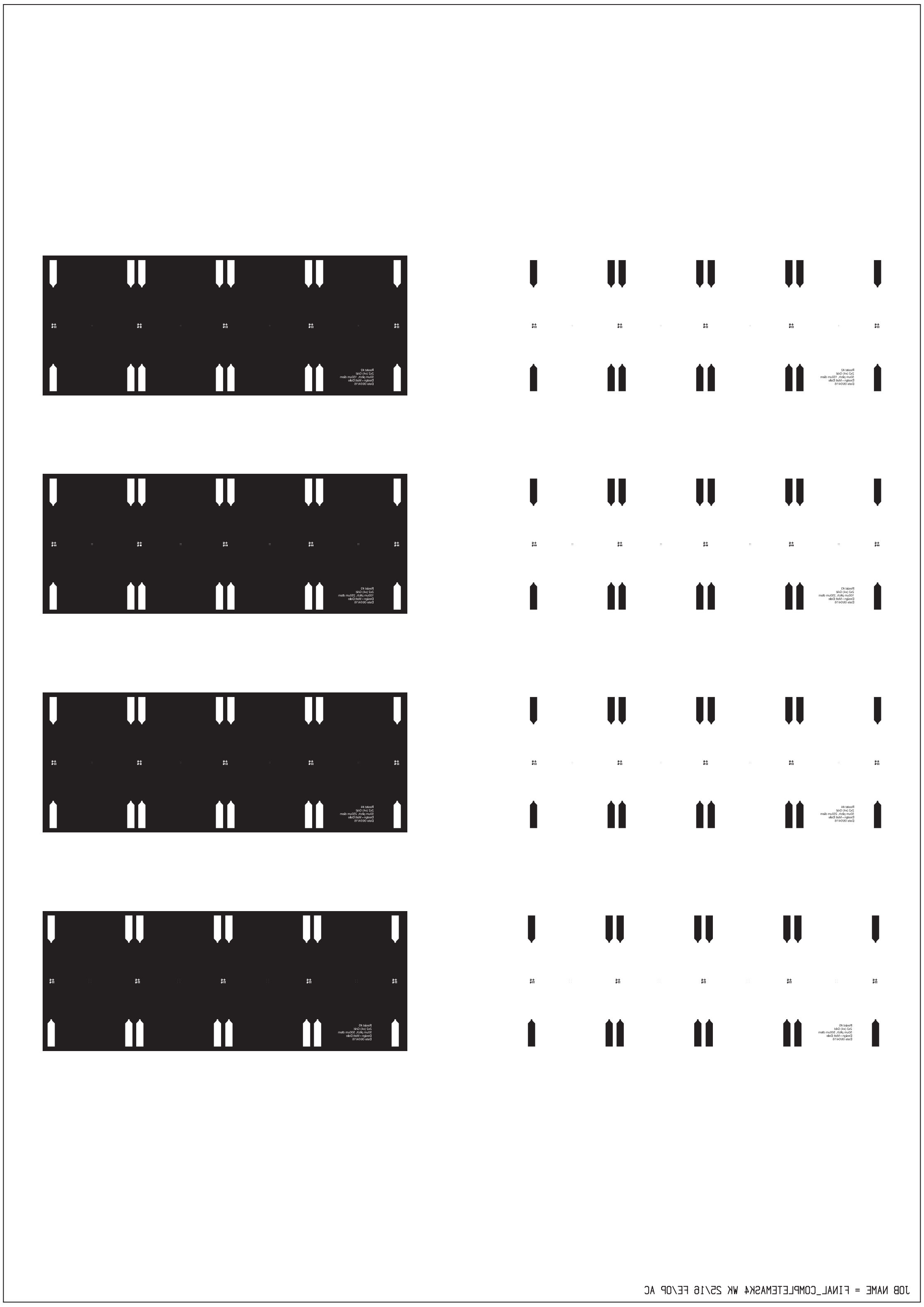

U45:A
U45:B
U45:C
U45:D
+12V
-12V
TL074
DAQ IN60
DAQ IN60B
DAQ IN61
DAQ IN61B
DAQ IN62
DAQ IN62B
DAQ IN63
DAQ IN63B
Buffers 3
FILE NAME: RevoMatMk2.pdsprj
DESIGN TITLE: RevoMatMk2.pdsprj
PATH: K:\Post Grad\D\Matt Dale\RevoMatMk2\RevoMatMk2.pdsprj
BY: @AUTHOR
REV: @REV
DATE: 15/03/2017
PAGE: 8 of 8
TIME: 13:57:57

A.3 Electrode Masks




JOB NAME = FINAL_COMPLETEMASKS2 WK 25/16 FE/OP AC


JOB NAME = FINAL_COMPLETEMASK3 WK 25/16 FE/OP AC

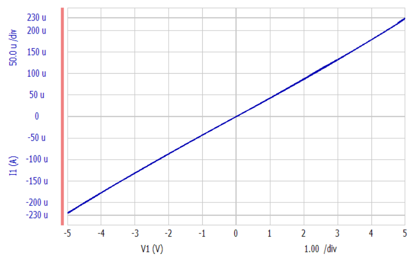

JOB NAME = FINAL_COMPLETEMASK4 WK 25/16 FE/OP AC

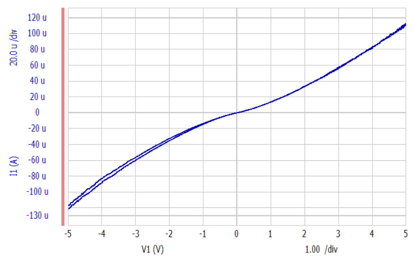
A.4 Substrate I-V Curves

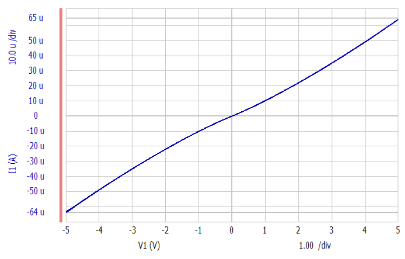
Sweeps from -5V to 5V @10mA

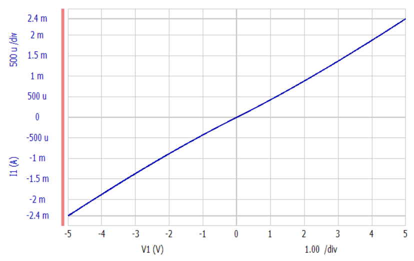
B02S06 – 0.71% PMMA

B15S11 – 1% PMMA



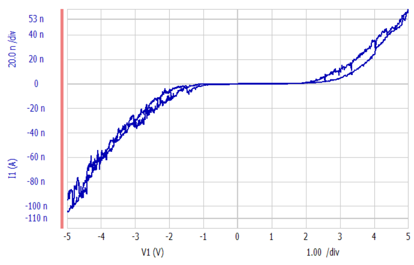


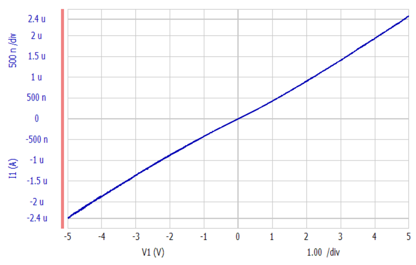


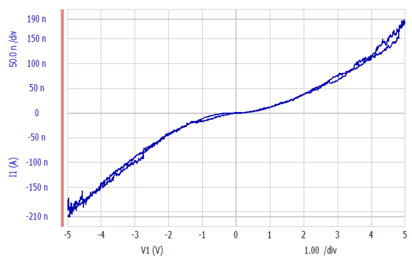


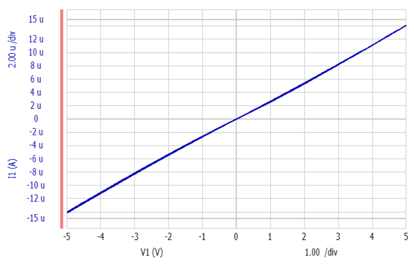
B16S02 – 0.02% PMMA

B16S03 – 0.032% PMMA



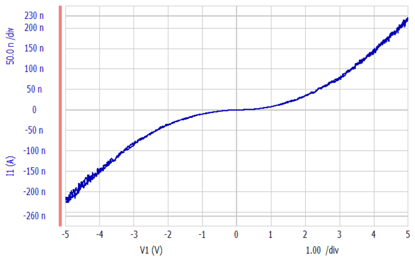


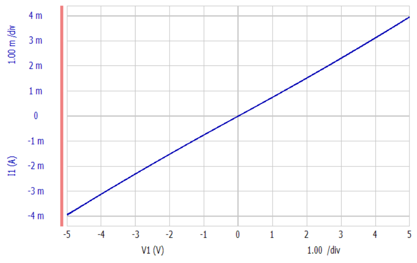


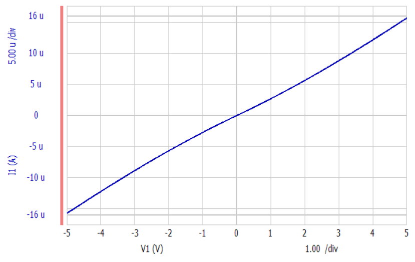


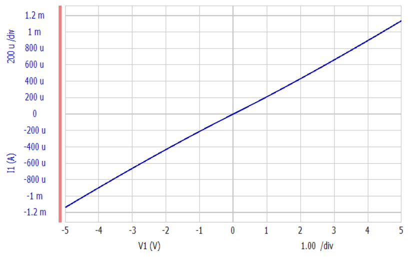
B16S04 – 0.042% PMMA

B19S01 – 0.71% PMMA





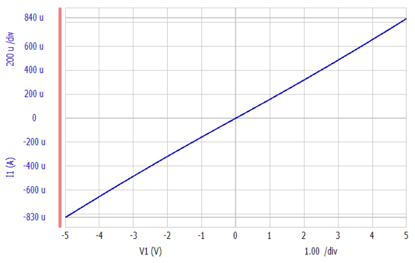


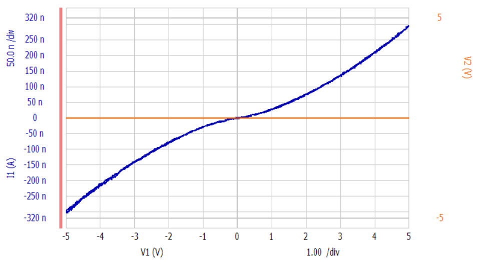


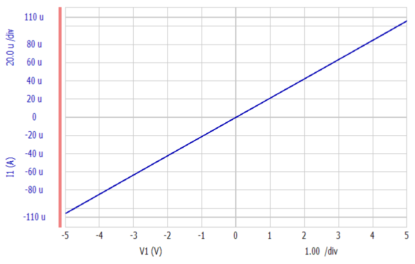
Sweeps from -5V to 5V @10mA (Bias added)

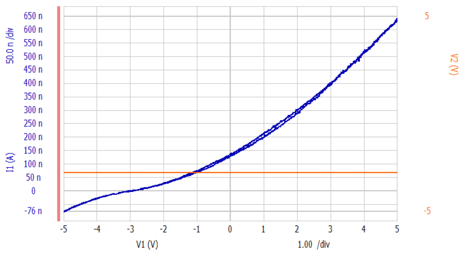
B19S02 – 0.5% PMMA

B12S11(spin coated) – 0.5% PBMA w/ 0V bias & -3V bias



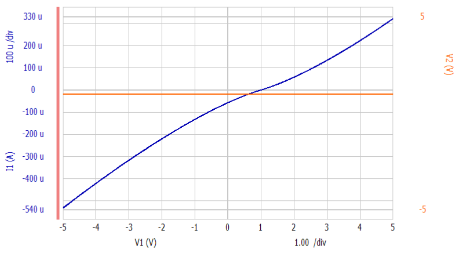


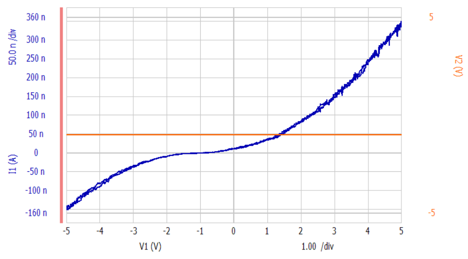


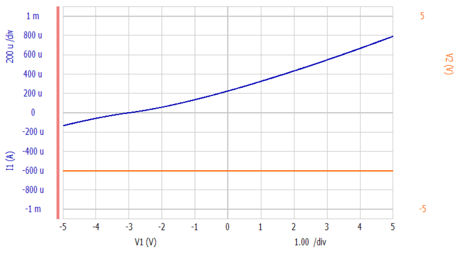


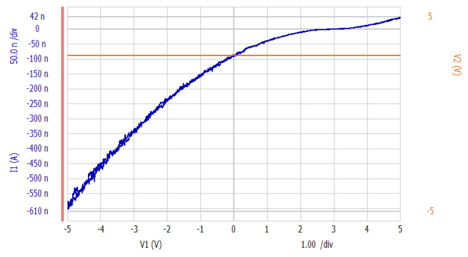
B15S12 – 1% PBMA w/ 1V bias,-3V bias & 5V bias

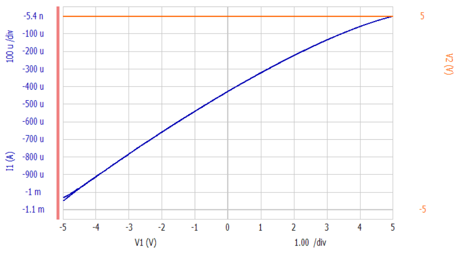
B16S04 – 0.042% PMMA w/ -1V bias, 3V bias & -5V bias

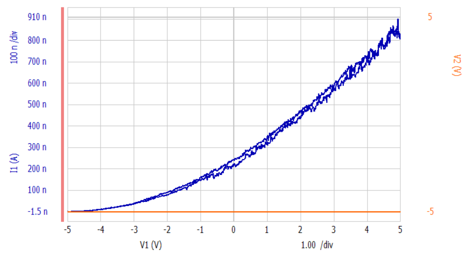








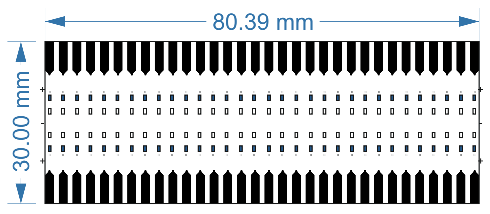




A.5 Multi-Substrate Housing and LED Designs

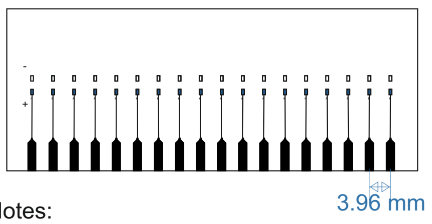
LED Test cards: Two-sizes, 64 and 16 test cards

32x2 LED Test card



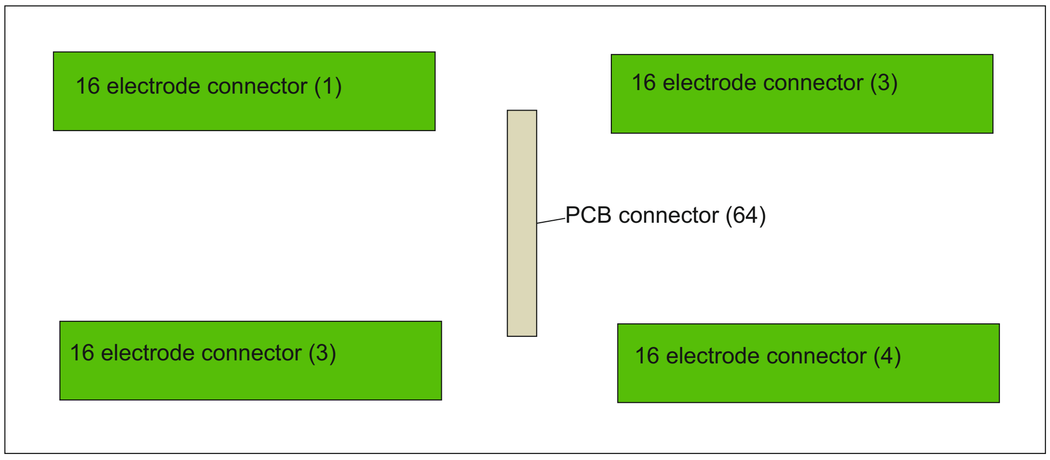
Notes:
- LEDs, e.g. 0402 SMDs (white) and resistors (black)
- Component placement has to leave room for card connector - about 8mm
- Dimensions: card insert 80mm x 30mm (at least, ref above). Pitch between contacts 2.54mm
- For EDAC 345-064-520-202 edge connector

16x1 LED Test card



Notes:
- Component placement has to leave room for card connector - about 8mm
- Dimensions: card insert 76mm x 30mm (at least, ref above). Pitch between contacts 3.96mm
- For EDAC 307-018-505-104 edge connector

Multi-connector board: 4x 16-pin edge connectors


16 electrode connector (1)
16 electrode connector (3)
PCB connector (64)
16 electrode connector (3)
16 electrode connector (4)

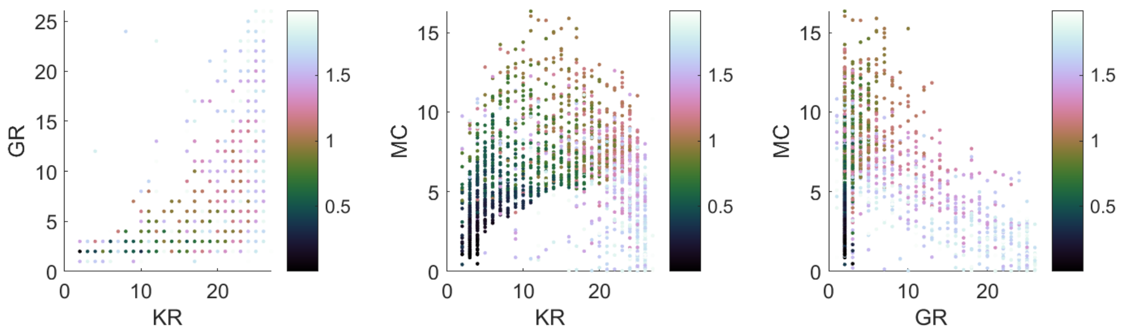
Notes:
- PCB connector is the same d-type 64 connector used on other board
- Electrode connectors are the EDAC 307-018-505-104 edge connector. Dimensions: 94mm x 8mm x 11mm
- No particular dimensions, but smaller the better.

Appendix B

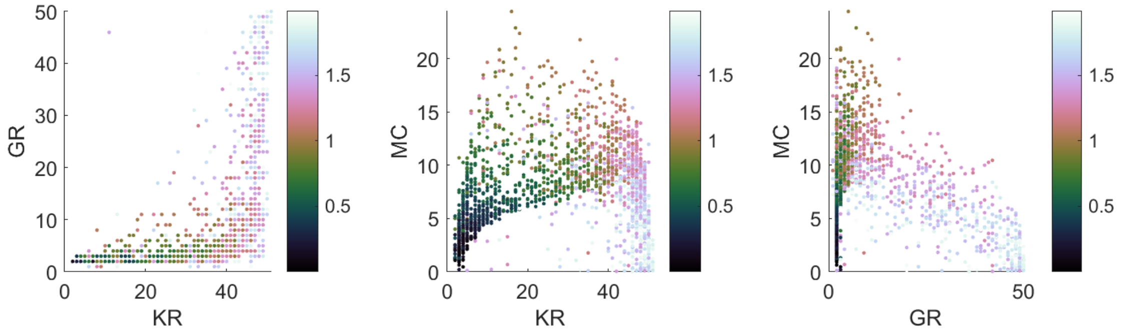
SQuARC: Supplementary ESN Plots

Here different plots are provided highlighting the relationships found between ESN parameters and the behaviour space. When conducting random search, certain ESN parameters, e.g. spectral radius and input scaling, have reoccurring relationships with the behaviour space across different ESN sizes (see Fig. B.1 and B.2). However, some are more difficult to interpret, e.g. leak rate and W connectivity (see Fig. B.3 and B.4).

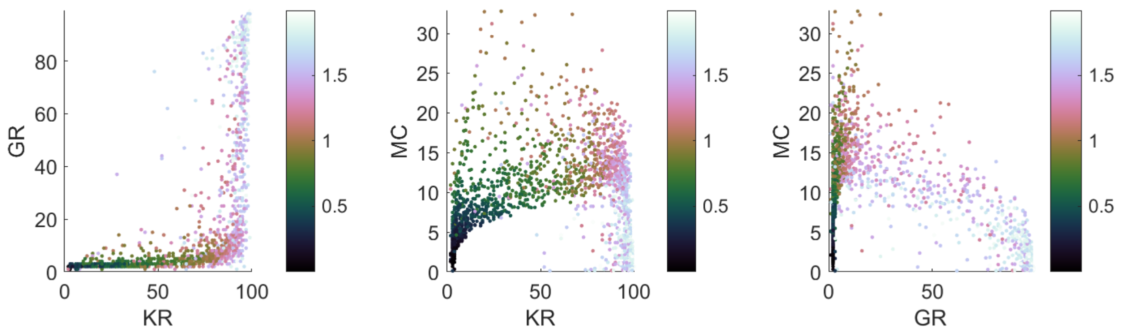
When using novelty search, the same relationships either breakdown or change. In many cases, much of the behaviour space is occupied around a localised parameter value, e.g. a spectral radius roughly around 1.2 in Fig. B.5. In other cases, relationships change depending on the network size, e.g. in Fig. B.6 (input scaling) and B.7 (leak rate). Much can still be learned about ESNs from these plots. For example, why novelty search appears to better exploit the connectivity of W (Fig. B.8), the leak rate (Fig. B.7), and the connectivity of the input (Fig. B.9).



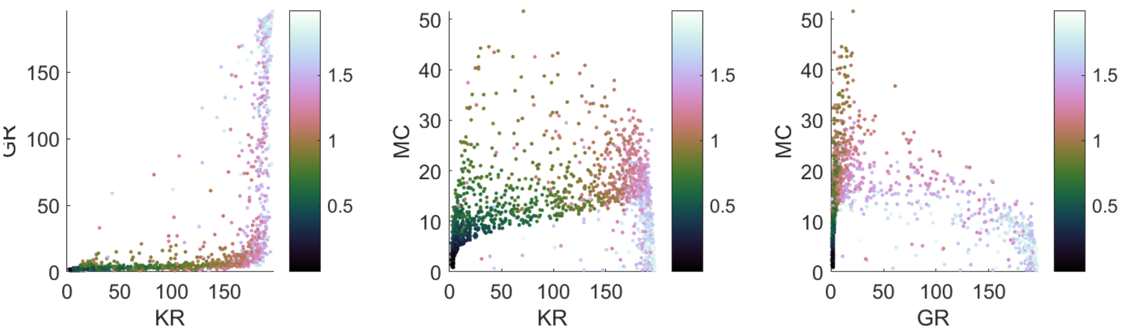
(a) 25 node



(b) 50 node

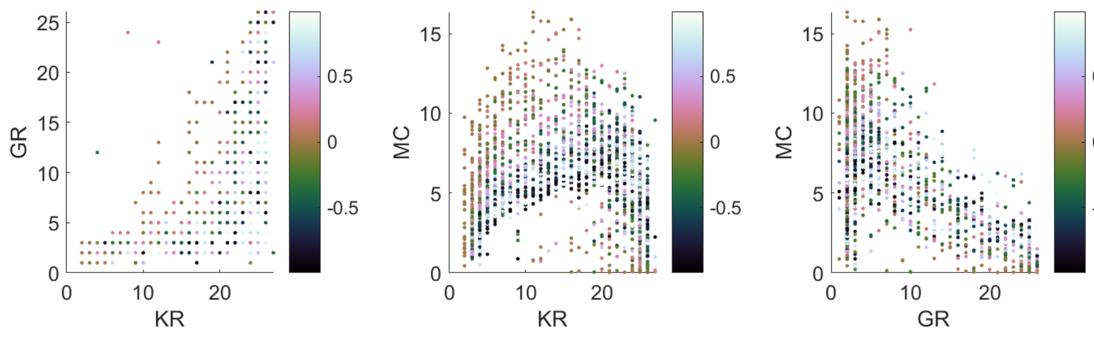


(c) 100 node

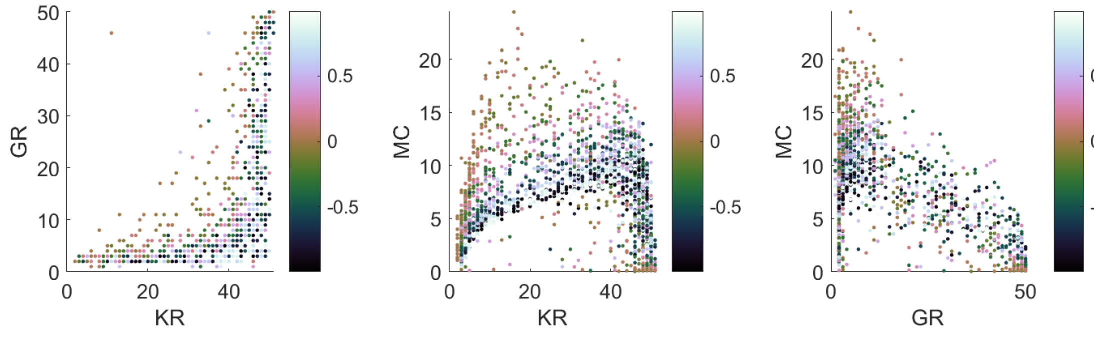


(d) 200 node

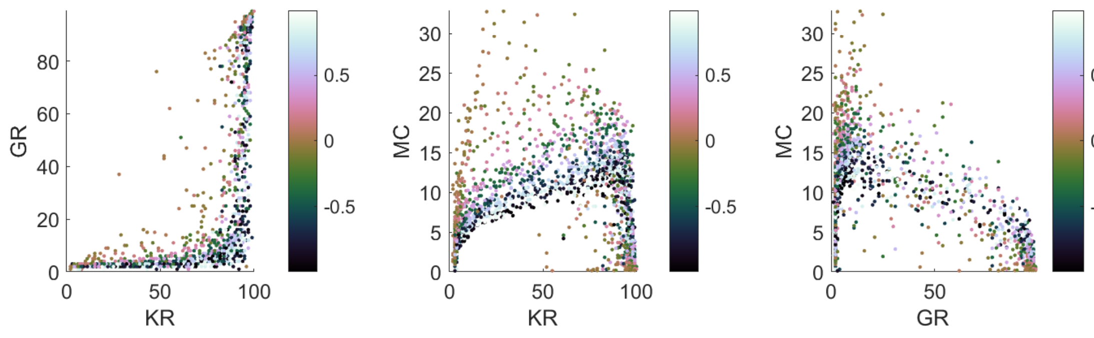
Figure B.1: Random Search: Spectral radius relationship to behaviour space.



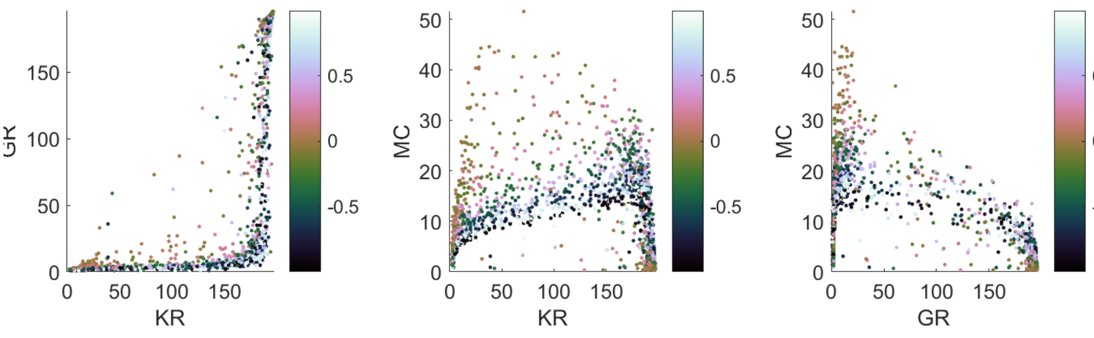
(a) 25 node



(b) 50 node

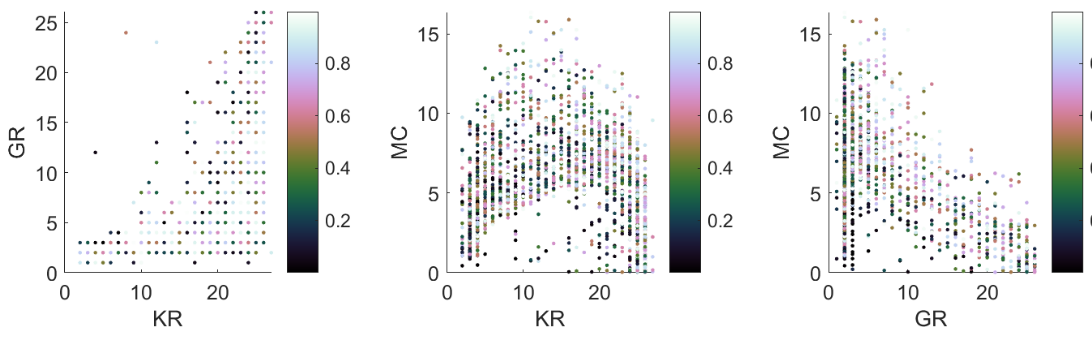


(c) 100 node

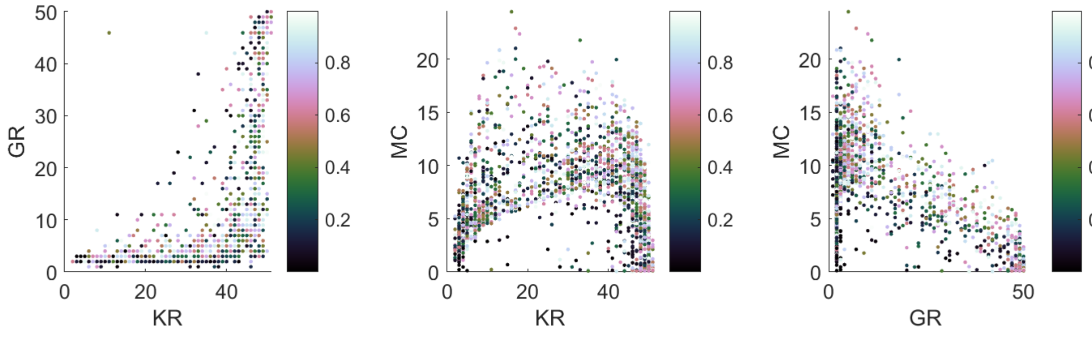


(d) 200 node

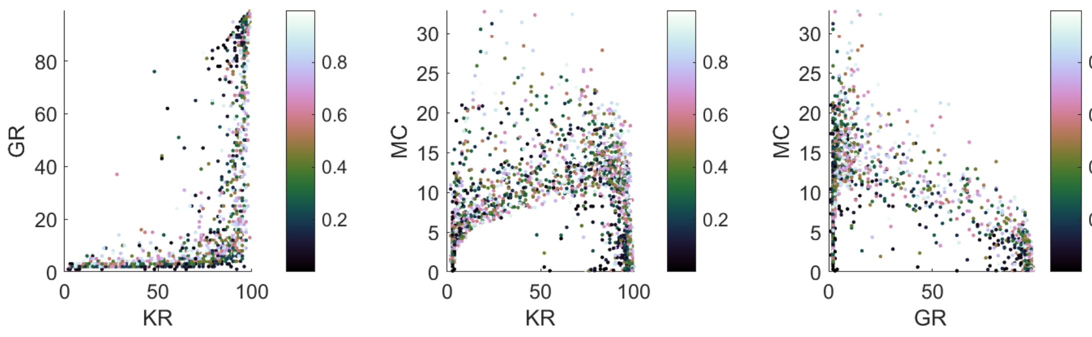
Figure B.2: Random Search: Input Scaling relationship to behaviour space.



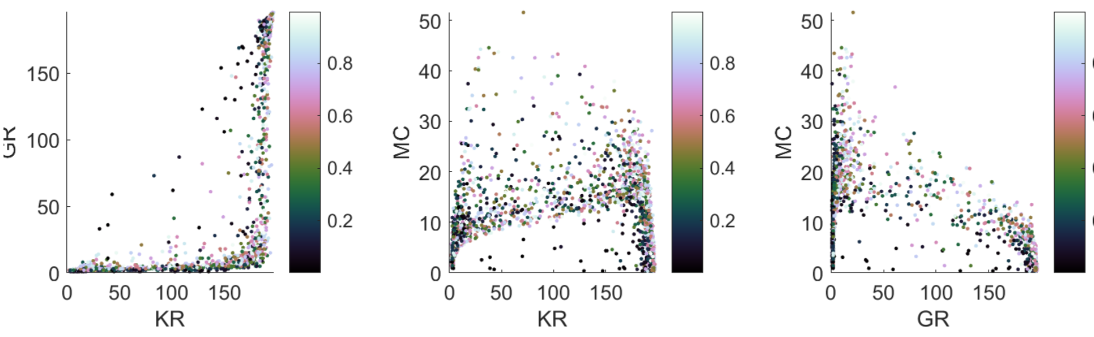
(a) 25 node



(b) 50 node

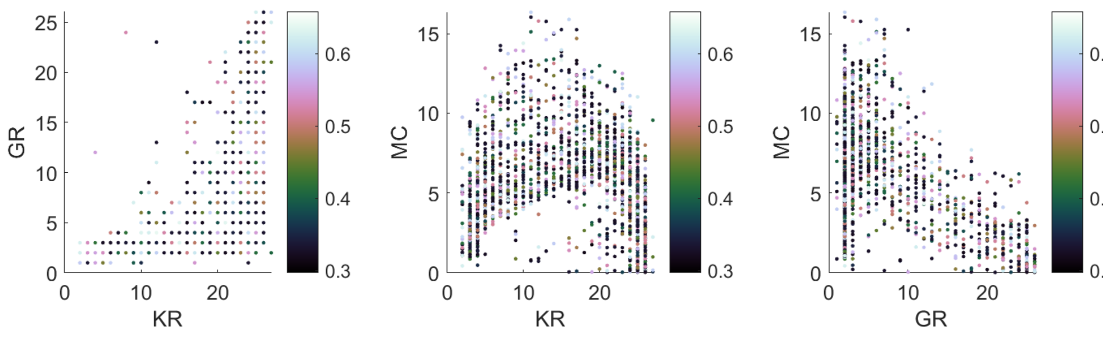


(c) 100 node

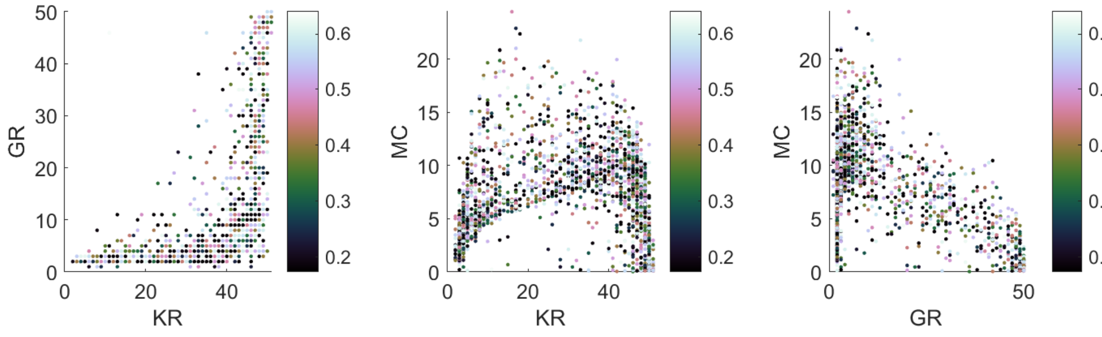


(d) 200 node

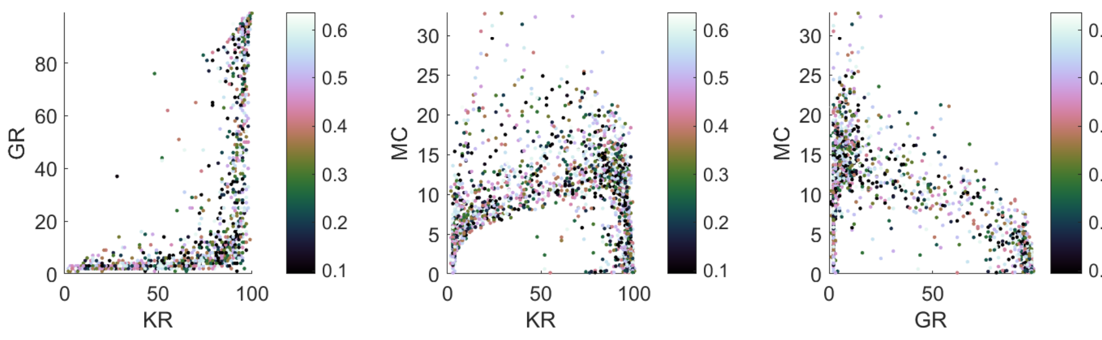
Figure B.3: Random Search: Leak rate relationship to behaviour space.



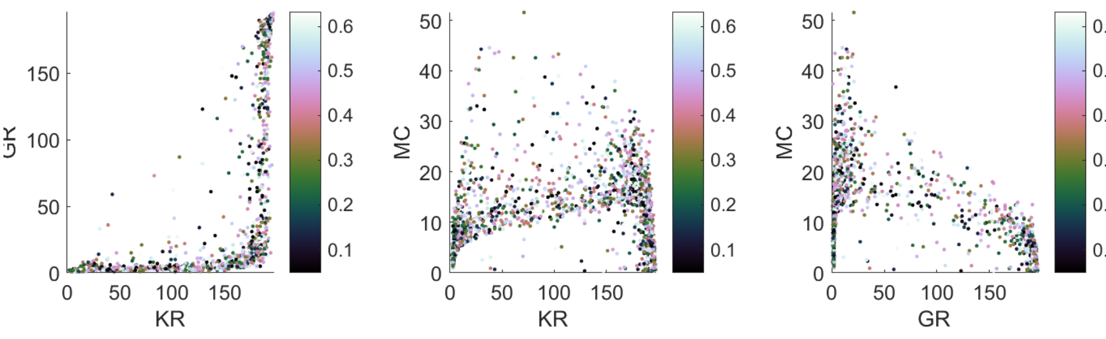
(a) 25 node



(b) 50 node

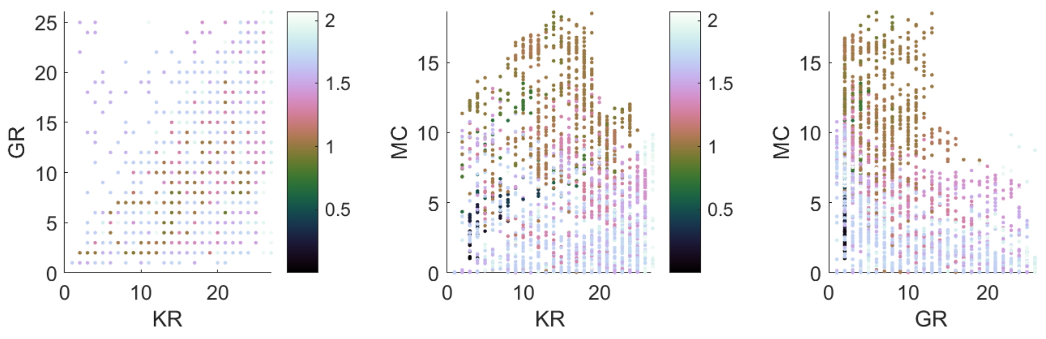


(c) 100 node

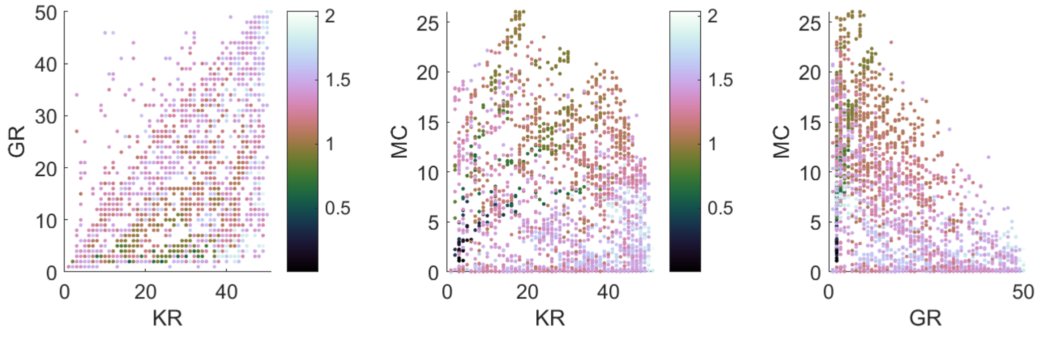


(d) 200 node

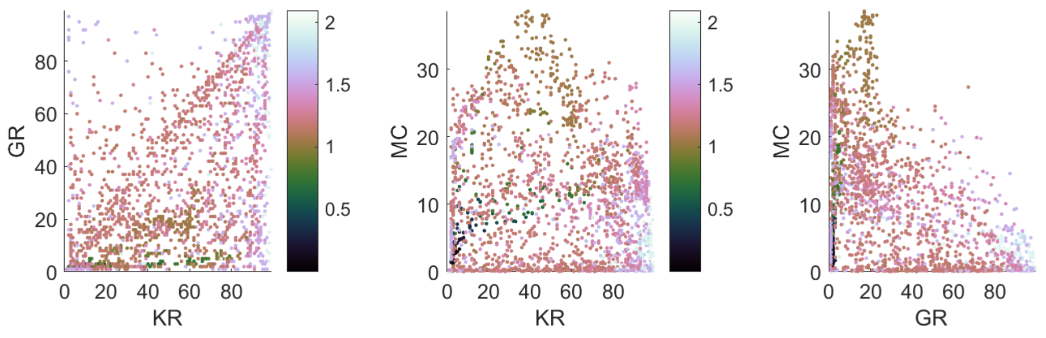
Figure B.4: Random Search: *W* connectivity relationship to behaviour space.



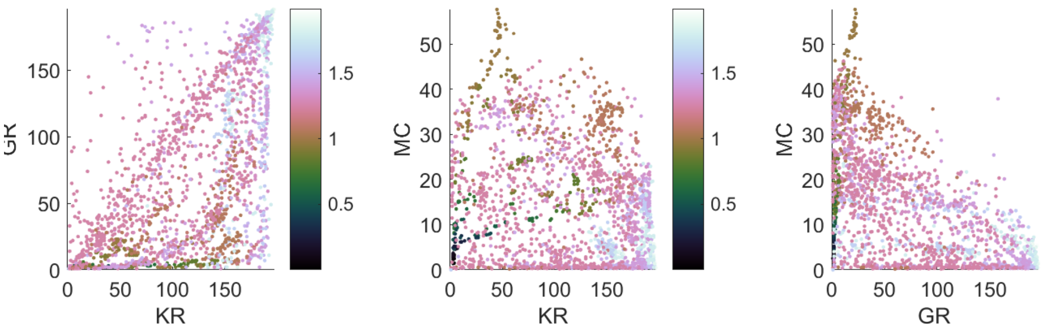
(a) 25 node



(b) 50 node

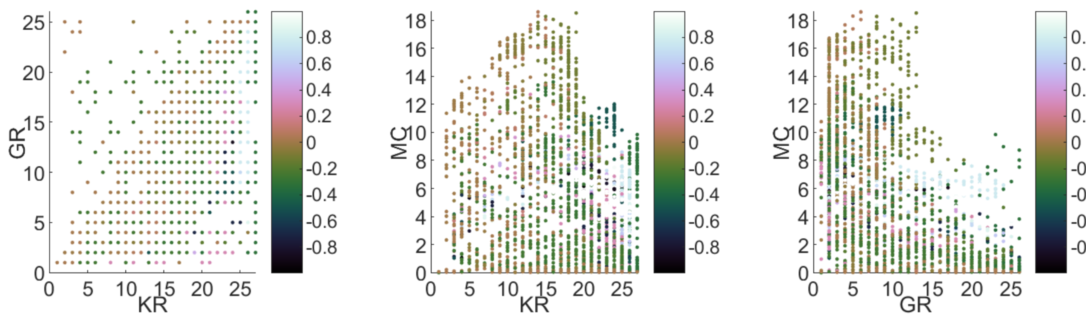


(c) 100 node

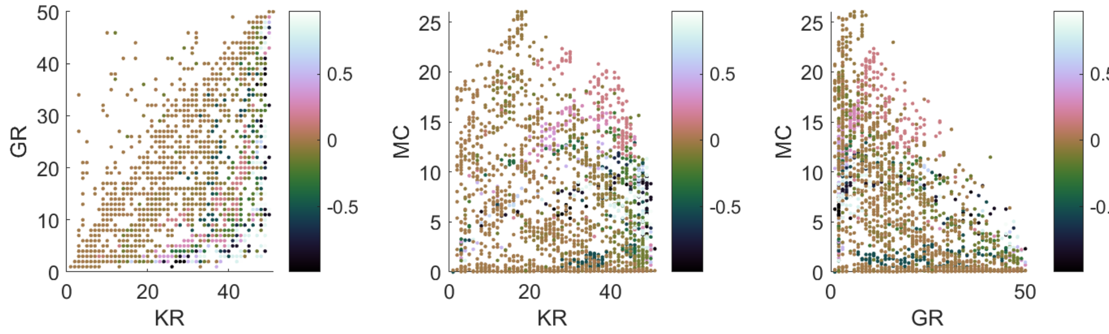


(d) 200 node

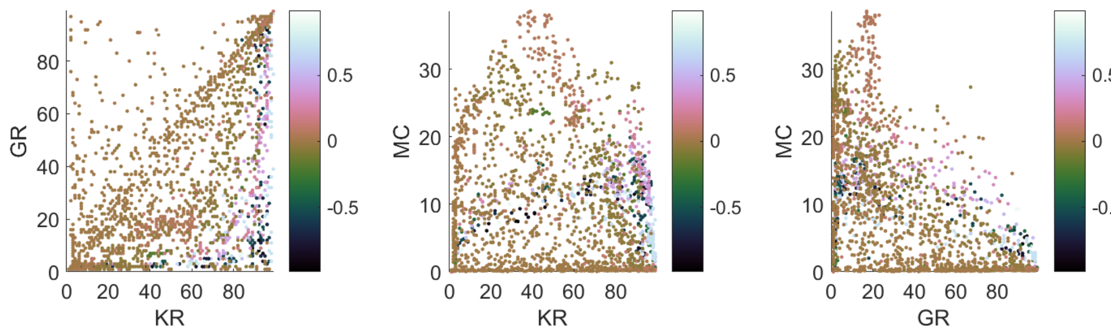
Figure B.5: Novelty Search: Spectral radius relationship to behaviour space.



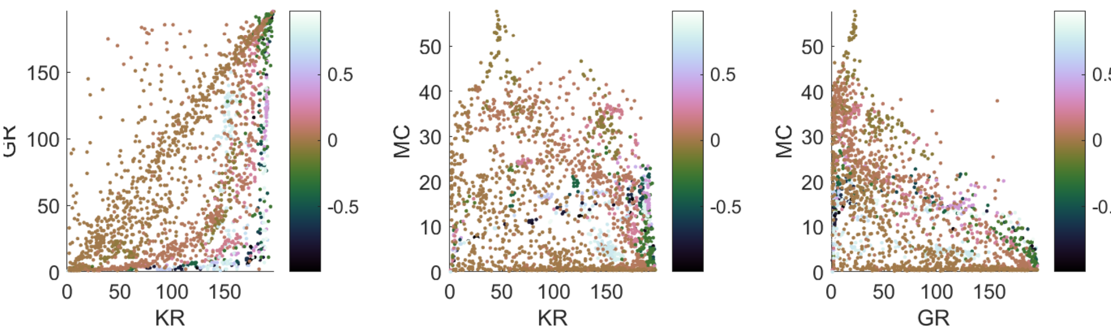
(a) 25 node



(b) 50 node

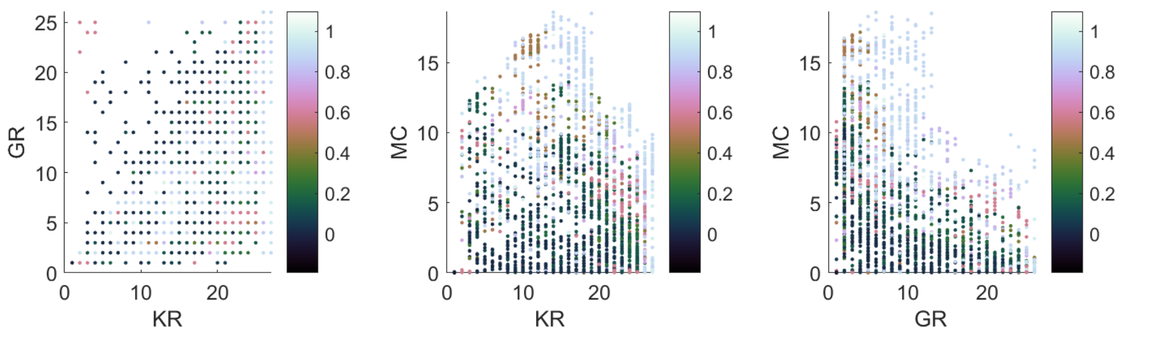


(c) 100 node

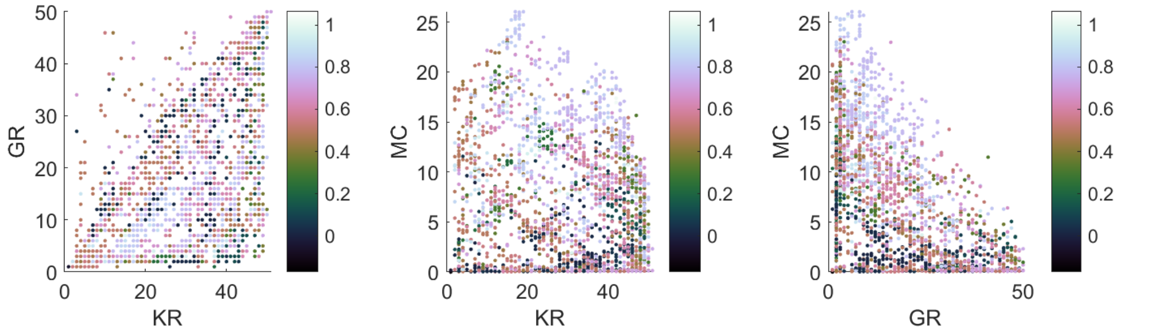


(d) 200 node

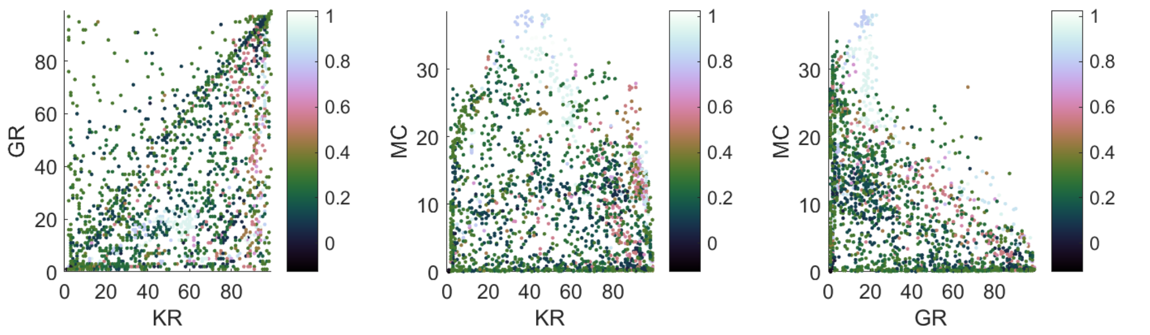
Figure B.6: Novelty Search: Input Scaling relationship to behaviour space.



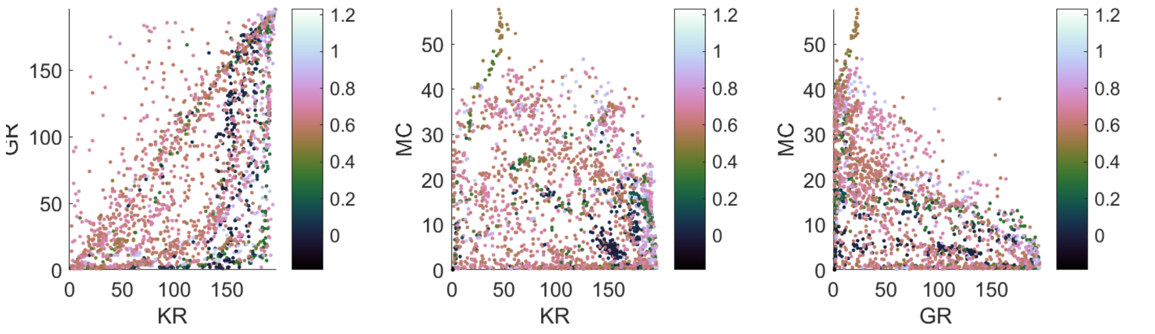
(a) 25 node



(b) 50 node

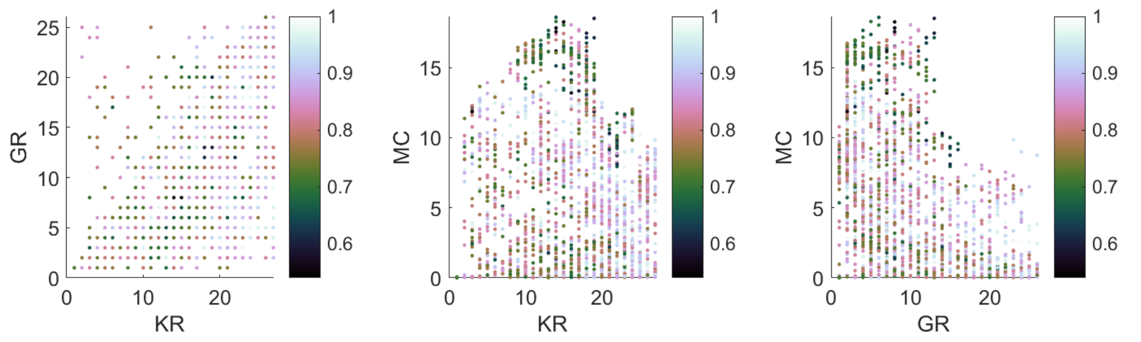


(c) 100 node

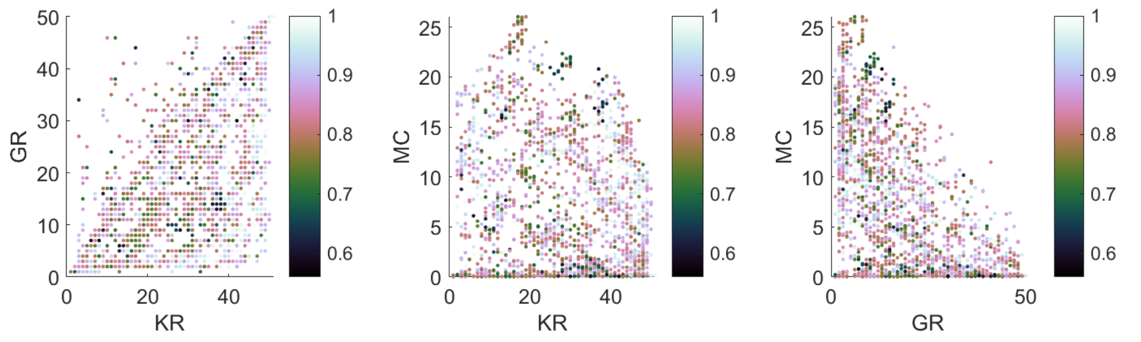


(d) 200 node

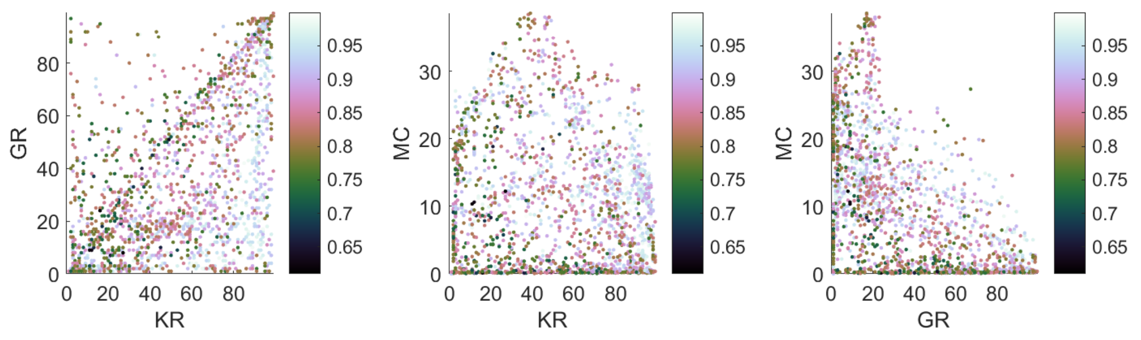
Figure B.7: Novelty Search: Leak rate relationship to behaviour space.



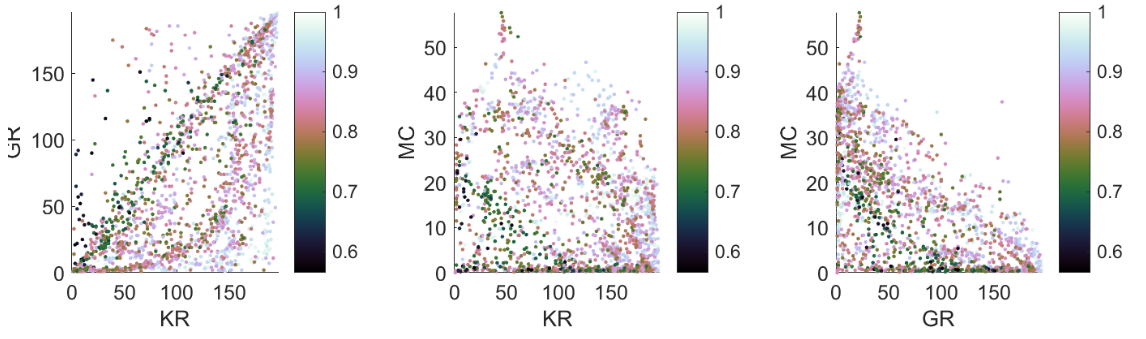
(a) 25 node



(b) 50 node

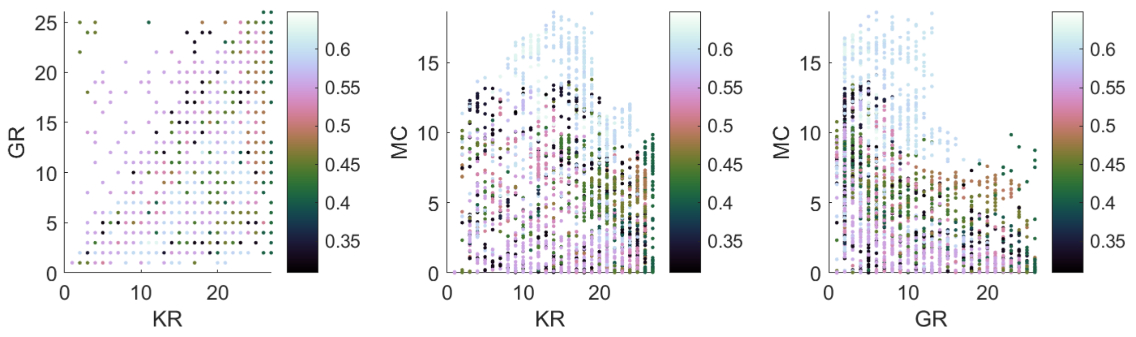


(c) 100 node

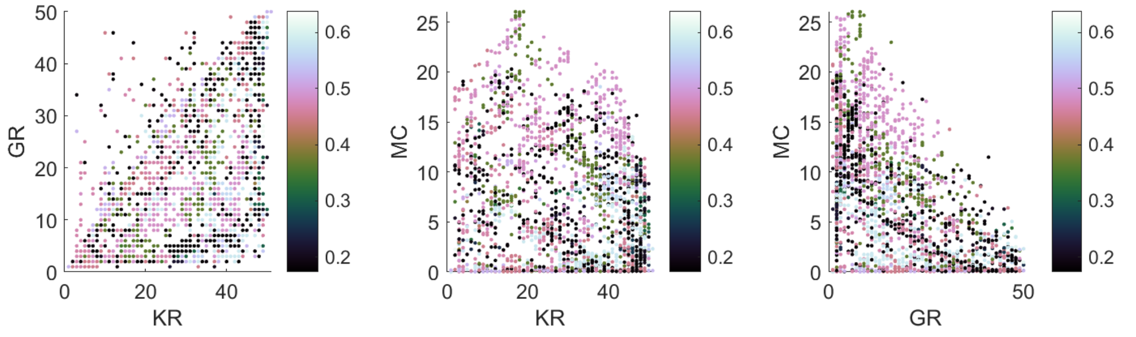


(d) 200 node

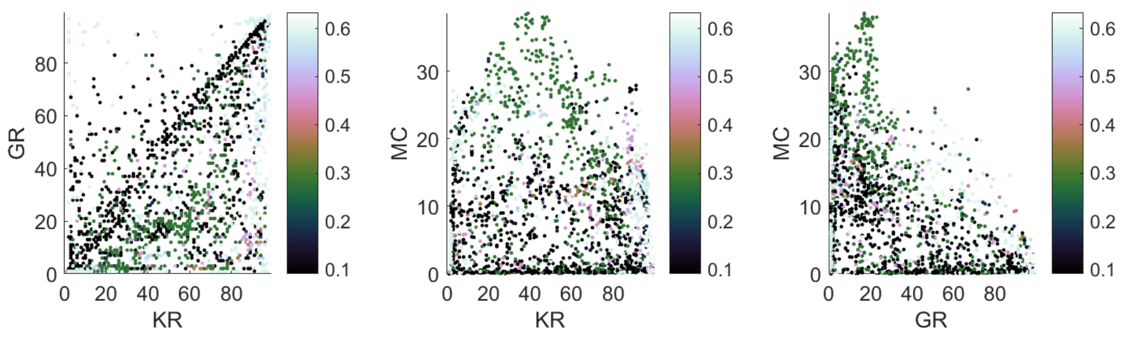
Figure B.8: Novelty Search: W connectivity relationship to behaviour space.



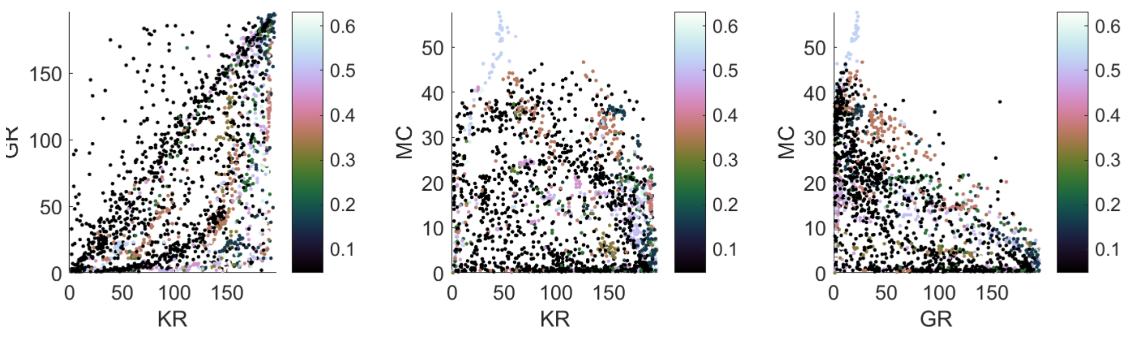
(a) 25 node



(b) 50 node



(c) 100 node



(d) 200 node

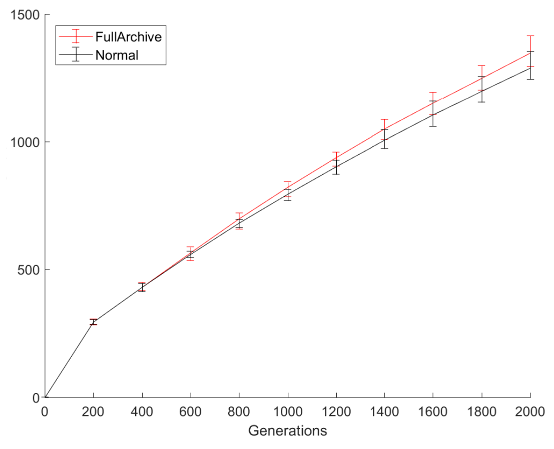
Figure B.9: Novelty Search: input connectivity relationship to behaviour space.

Appendix C

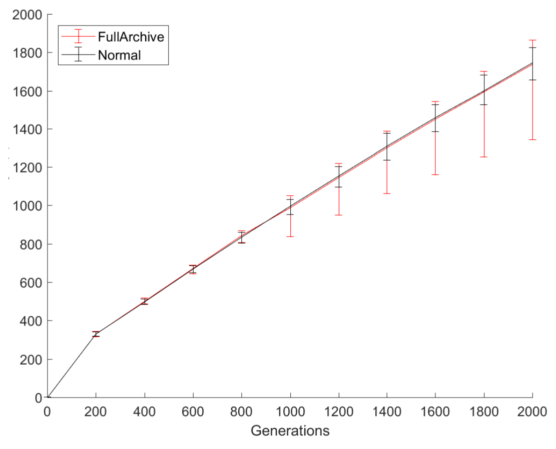
Dynamic Archive or All Behaviours?

As a side experiment, I thought it would be interesting to see what effect adding all the behaviours to the archive would have. Therefore, fitness would be assessed based on all previously explored behaviours. In practice, this results in a very large archive being stored. However, selection pressure may be stronger to move away from previously explored regions.

To test this, two network sizes we used; 25 and 50 node ESNs. According to the results (Fig. C.1 and C.2), adding all behaviours leads to mixed gains and losses in coverage. For the 25 node it may look like an advantage, however, across 10 runs it does not appear to be significant. Similar results are seen at 50 nodes, despite lower values in a few runs, there is no significant difference.

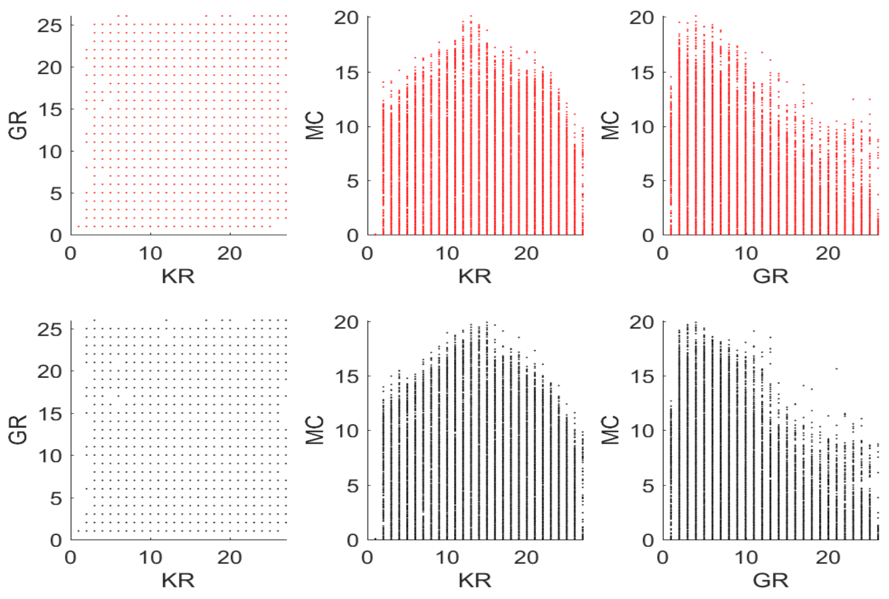


(a) 25 node

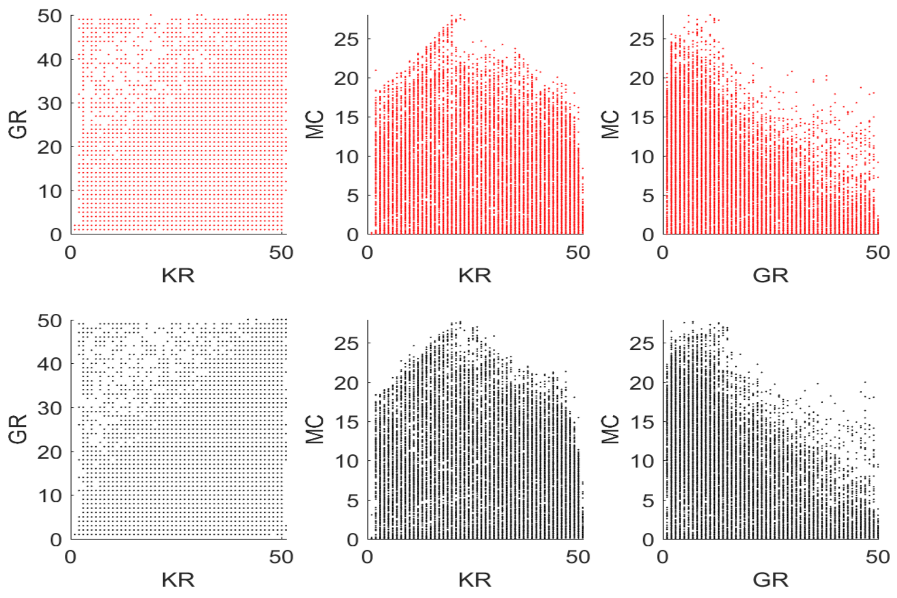


(b) 50 node

Figure C.1: All behaviours in the archive versus a dynamic sized archive: behaviour space coverage against number of generations.



(a) 25 node



(b) 50 node

Figure C.2: All behaviours found with “full” archive (Red, Top row) and all behaviours found with dynamic archive (Black, Bottom row).

Appendix D

Multi-Thread SQuARC Framework

For the SQuARC framework, implemented with ESNs, a multi-thread version was also built. The aim was to utilise the asynchronous capability of the microbial GA.

In order to parallelise the algorithm a number of adjustments were made. Each thread was set to perform selection on the same population and create its own offspring. Then, the output/offspring of each thread was collected and compared to see if any individual was concluded the loser multiple times. If any were to lose more than once, a comparison check was performed to see which new offspring produced the higher fitness, i.e. in a sparser region of the behaviour space. The winning offspring was then selected to supersede the losing individual in the population and thus carried forward to the next generation. Any losing offspring however was still added to the database, along with all other offspring created.

Having a parallel version of the algorithm allows faster and greater exploration of the space, given roughly the same time to compute, as seen in Fig. D.1 and D.2a. However, the asynchronous update of the population can produce other effects. For example, more exploitation than exploration, resulting in longer searches within a niche. To test this potential side-effect, the serialised and parallel implementations were compared given the same database size; as the database indicates how many reservoirs have been created/bred and thus evaluated.

The resulting behaviours of both searches are shown in Fig. D.2. The (black) bottom plots in Fig. D.2a show the serial implementation of a 200 node ESN, with a database size of 2000. Fig. D.2b shows the parallel implementation also with a database size of

2000. In the parallel version, the behaviours appear to less uniformly cover the space, suggesting greater exploitation.

To quantify whether greater exploitation is occurring, distribution plots of each dimension are given in Fig. D.3. In particular, kernel rank appears to concentrate more compared to the non-parallel equivalent. This could amount to more local search in the kernel rank dimension, or more likely, more chaotic reservoirs being produced; as indicated by the increase in high generalisation rank (Fig. D.3a). So rather than more exploitation, it could just suggest more unstable reservoirs are created using the parallel implementation.

In general, this side-effect might be offset by the vast numbers of behaviours searched within the same time-frame. This suggests room for improvement, however, the parallel implementation is likely to be preferred over the original.

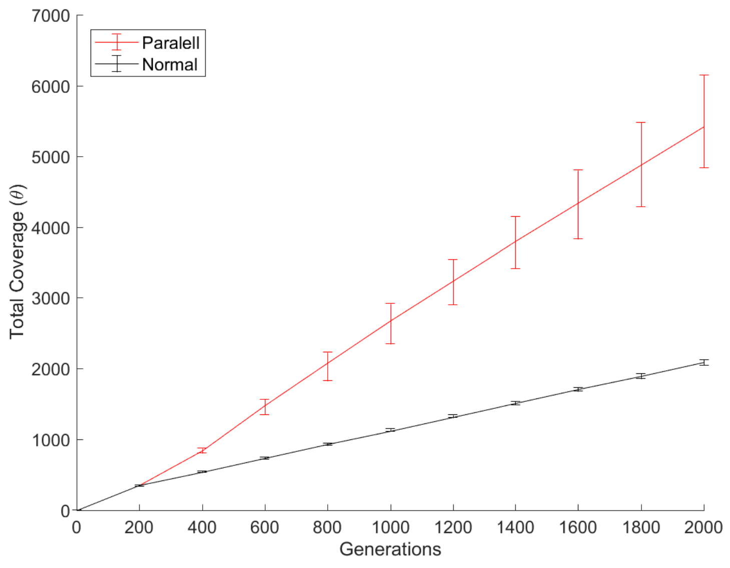
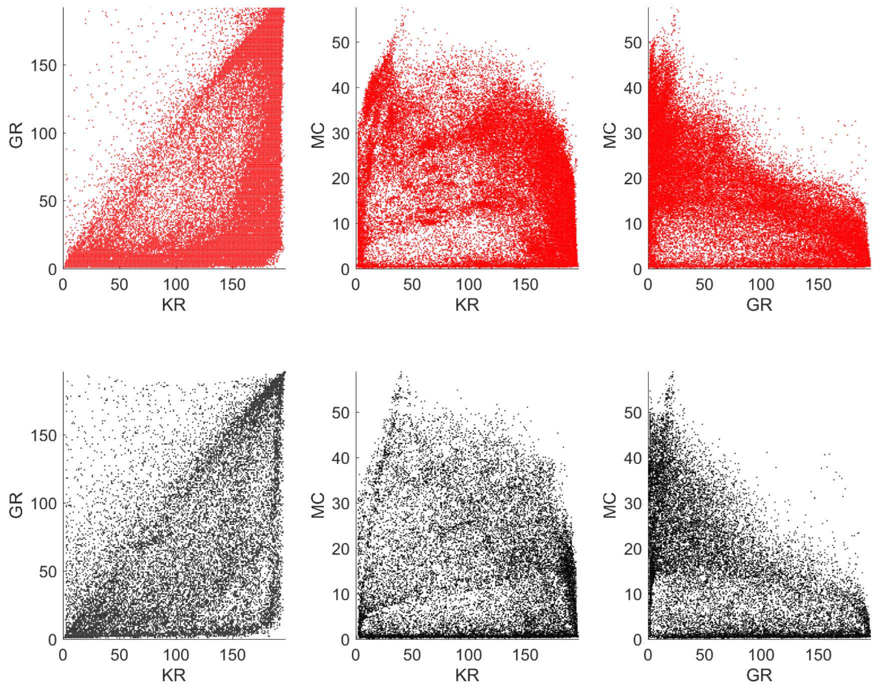
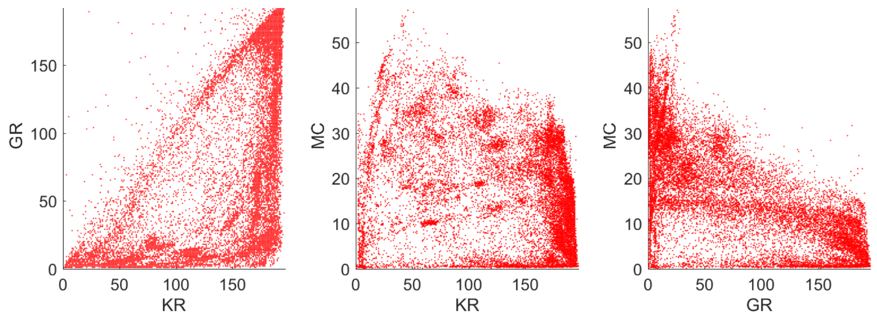


Figure D.1: Total coverage against generations: Parallel versus normal implementation.

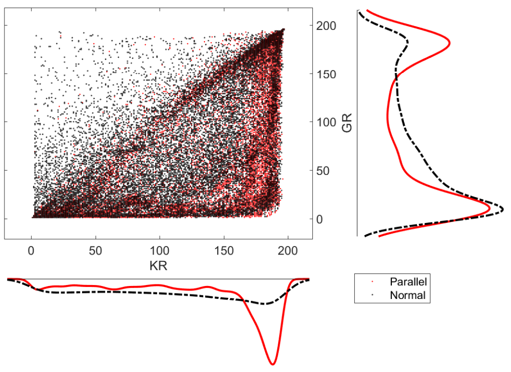


(a) Parallel vs. Normal

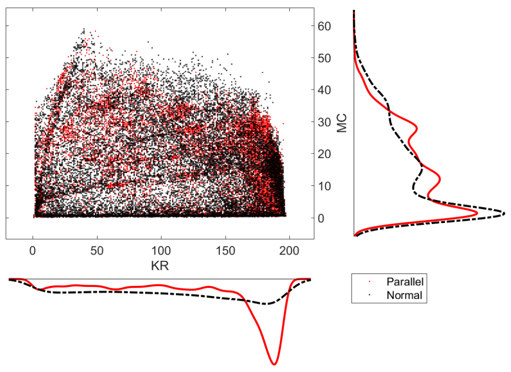


(b) Parallel with equal database size

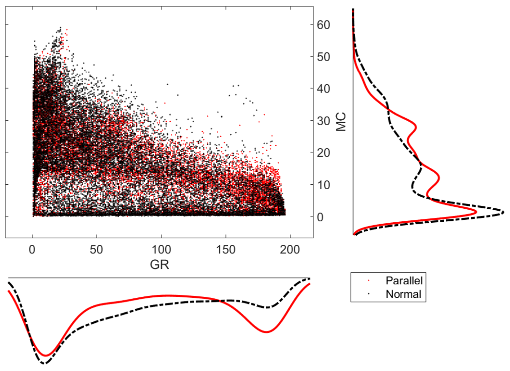
Figure D.2: a) All behaviours found with parallel implementation (Red, Top row) and all behaviours found with normal implementation (Black, Bottom row). b) Behaviours of parallel when set to equal database size.



(a) KR vs. GR Distribution



(b) KR vs. MC Distribution



(c) MC vs. GR Distribution

Figure D.3: Comparison of behaviour distributions: Parallel versus Normal, with same database size.

Appendix E

SQuARC: Supplementary Prediction Plots

Two learning systems were investigated as part of the SQuARC "learning" level, using all the collected ESN data to solve the behaviour space/task performance regression problem. These were Bagged (Ensemble) Decision Trees and Feed-Forward Neural Networks (FFNN).

In Fig. E.1 to E.8, results of each prediction model are shown for each ESN size. Each trained separately on a task, then tested with every ESN size. To represent this, heat maps are given, where trained size y (y axis) is evaluated on test size x (x axis). Every ESN size is modelled and tested 10 times, with the average prediction error of the model (Root Mean Square Error (RMSE) and Mean Absolute Error (MEA)) displayed.

Each figure represents one of four data configurations; using *three* (original KR, GR and MC) or *eleven* (functions of the originals) features, with a task performance threshold/ceiling set to 1 or 0.8. A threshold/ceiling was set to remove any data points outside the set value – in some cases the target data (ESN task error) was very large, resulting in large bias in the training stage.

In general, the two techniques used to predict ESN performance do well. They also tend to generalise better when the model is trained on data from a larger ESN than the test ESN; in all cases for the FNN model, but not the bagged decision trees.

The task that appears to be the easiest to learn is the NARMA-30 task and the hardest is the non-linear channel equalisation task.

The most accurate predictions are seen using a ESN task error threshold of 0.8. The number of features, however, affect the prediction less.

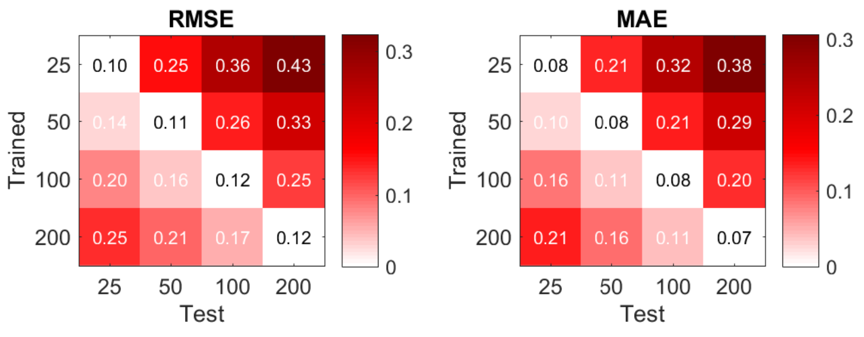
Applying the best settings: 11 features and a threshold of 0.8, both the bagged decision trees and feed-forward networks perform roughly the same. The decision trees generalise better than the FFNNs when the model is trained on a smaller ESN size and tested on a larger ESN size, however generalise slightly worse in the opposite configuration. Based on these results, out of the two, feed-forward networks are recommended as generalisation with $y \geq x$ is slightly better.

In a second experiment, a new FFNN was trained to predict all tasks simultaneously (see Fig. E.9). As with previous tests, multiple data configurations were set.

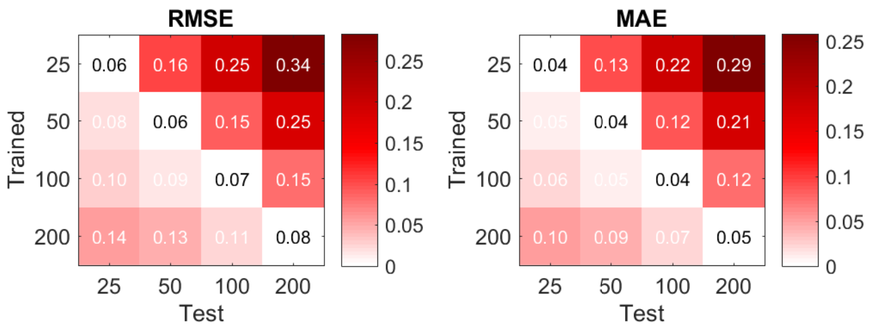
The results of the multi-task predictor suggest a feed-forward network can learn and predict all tasks well; this advantage requires fewer networks to be trained per task. In terms of configuration, a threshold of 0.8 and using 3 features tends to work best.

In the final experiment, four FFNNs were trained on all the data collected (per task), i.e. the databases from all ESN sizes. The objective was to see how well the FFNNs generalise to each ESN size when trained on everything.

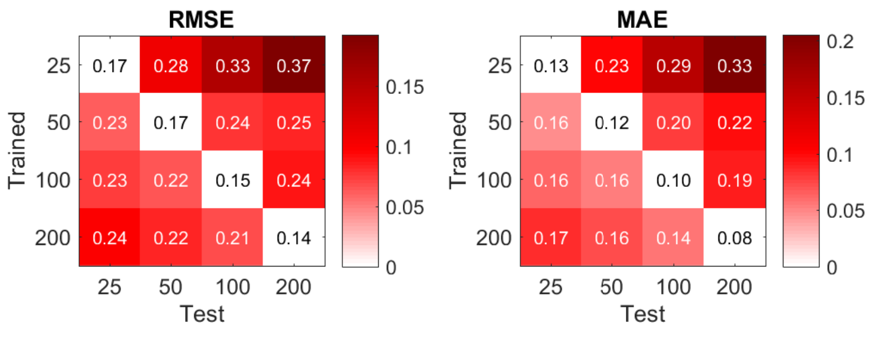
The results are shown in Fig. E.10. The networks perform similarly to the networks trained in Fig. E.6, suggesting multiple substrates can be used to improve the accuracy and range of the predictive systems.



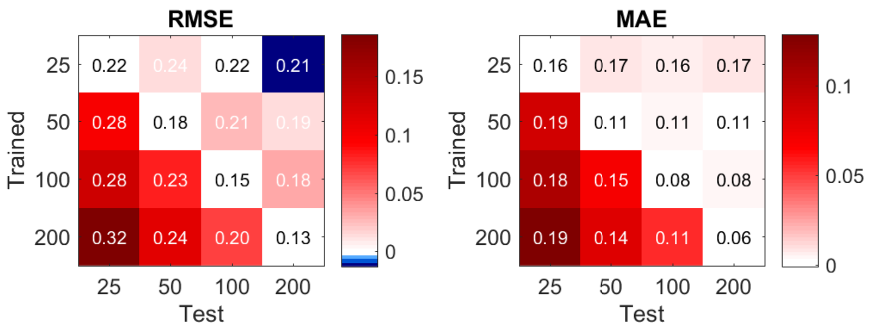
(a) NARMA-10



(b) NARMA-30

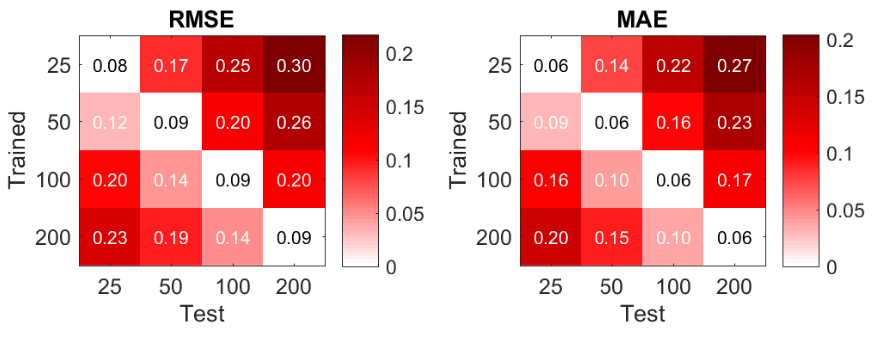


(c) Laser

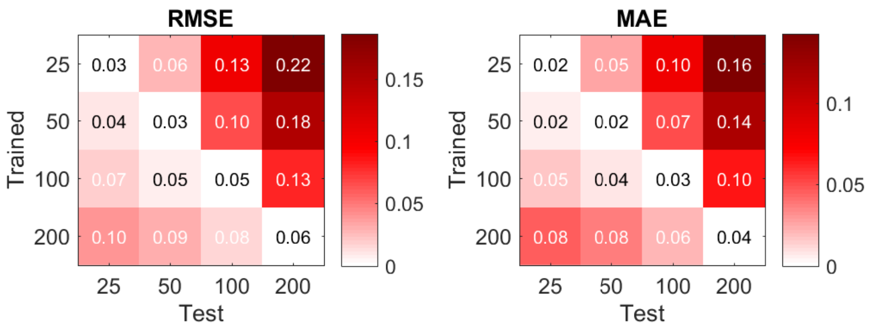


(d) Non-linear Channel Equalisation

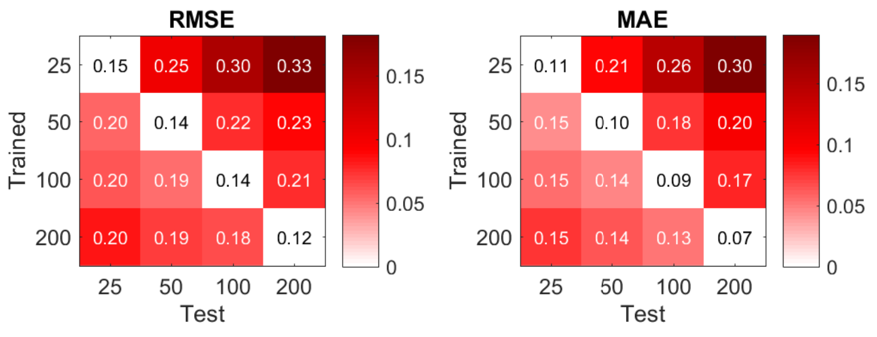
Figure E.1: Bagged Trees: Features=3, Threshold=1.



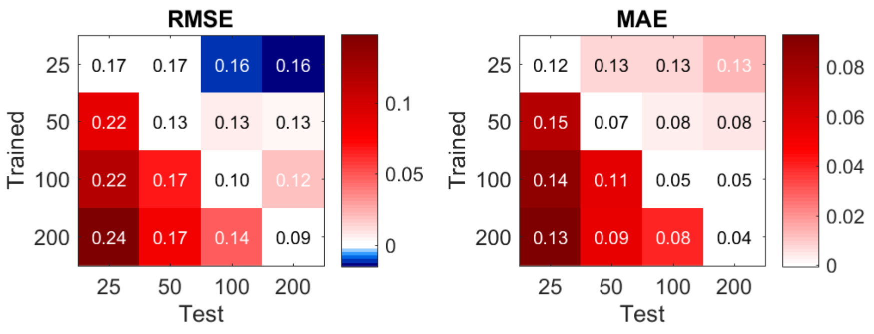
(a) NARMA-10



(b) NARMA-30

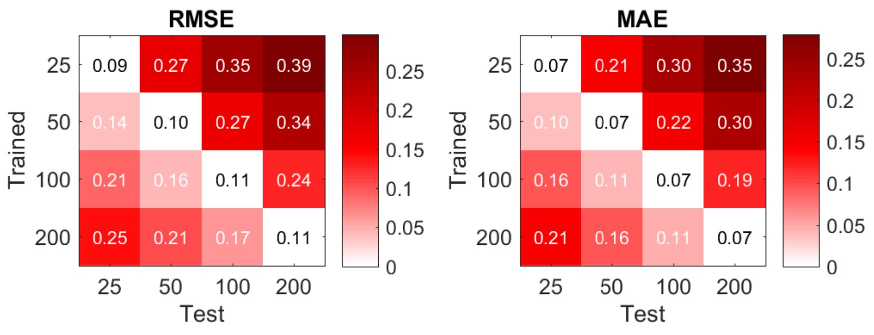


(c) Laser

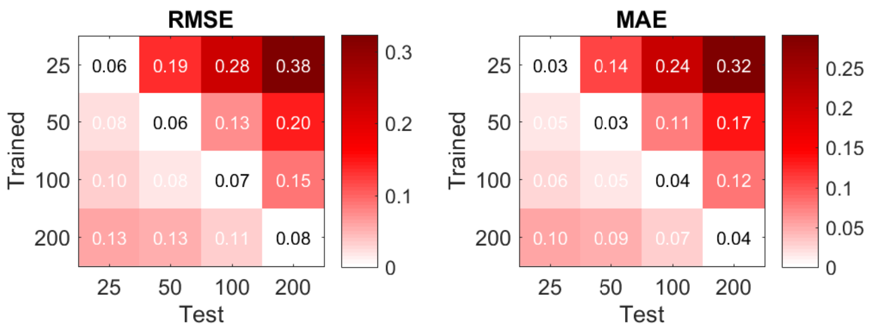


(d) Non-linear Channel Equalisation

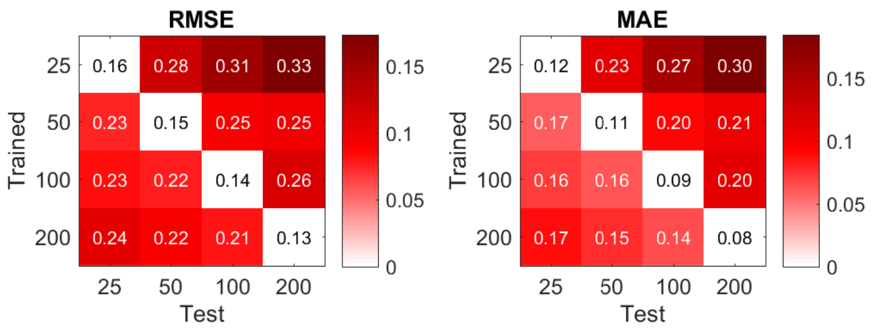
Figure E.2: Bagged Trees: Features=3, Threshold=0.8.



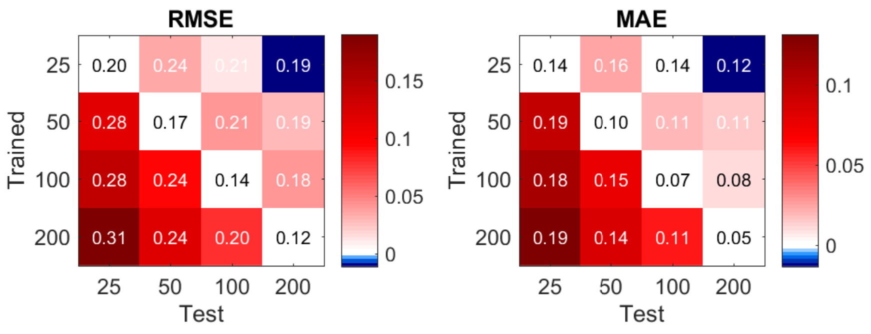
(a) NARMA-10



(b) NARMA-30

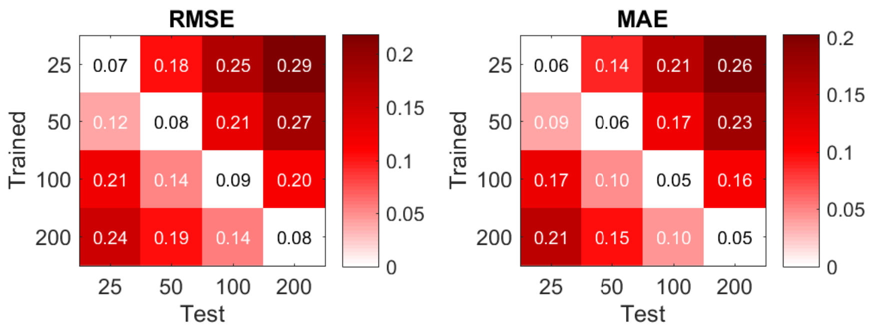


(c) Laser

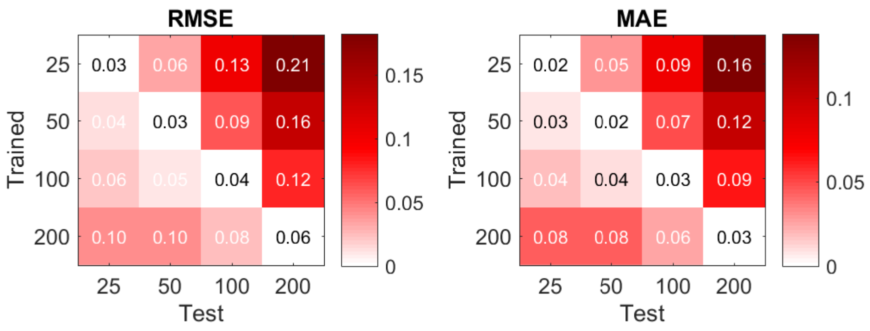


(d) Non-linear Channel Equalisation

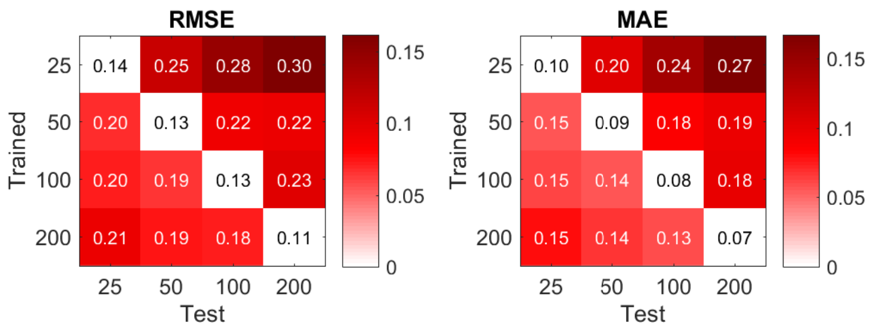
Figure E.3: Bagged Trees: Features=11, Threshold=1.



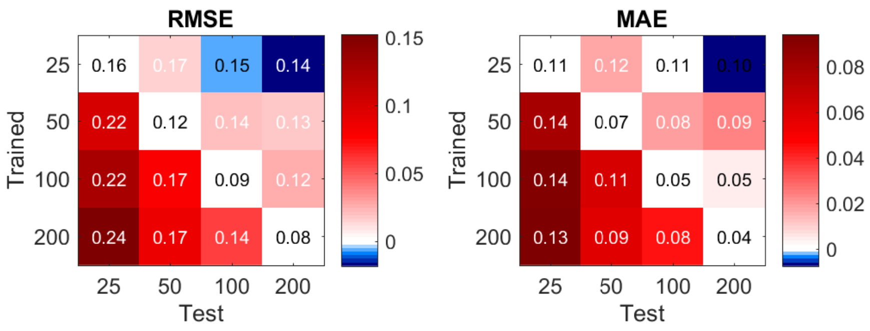
(a) NARMA-10



(b) NARMA-30

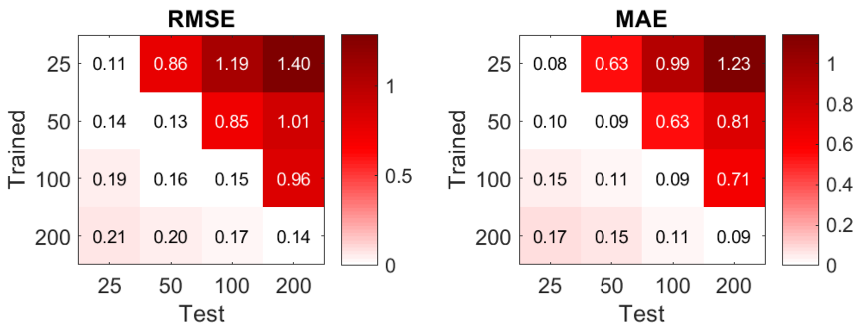


(c) Laser

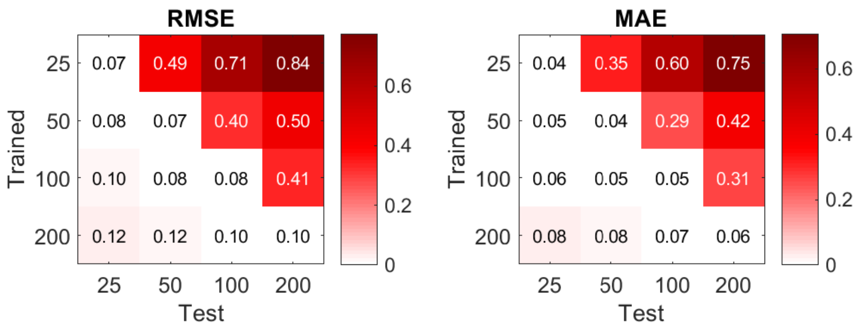


(d) Non-linear Channel Equalisation

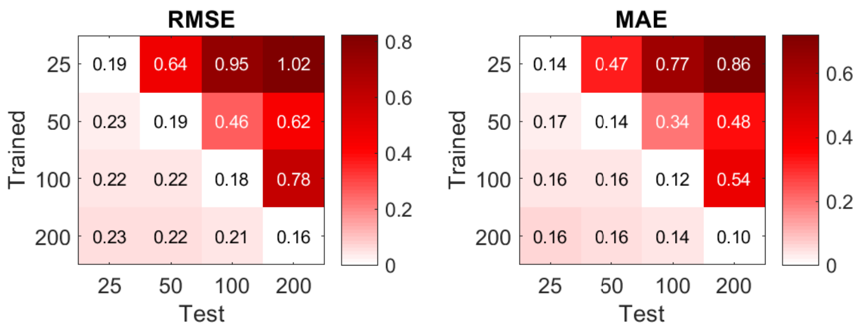
Figure E.4: Bagged Trees: Features=11, Threshold=0.8.



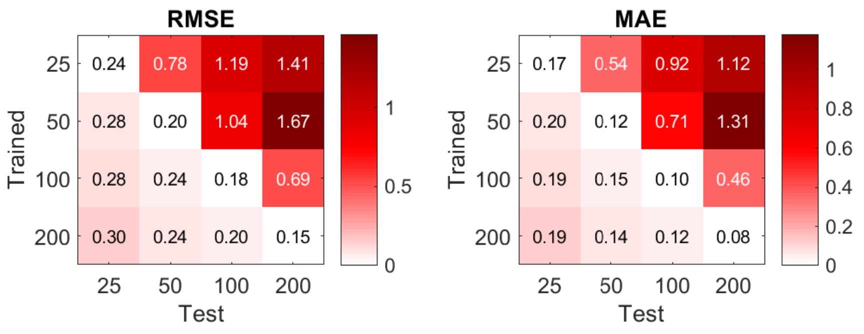
(a) NARMA-10



(b) NARMA-30

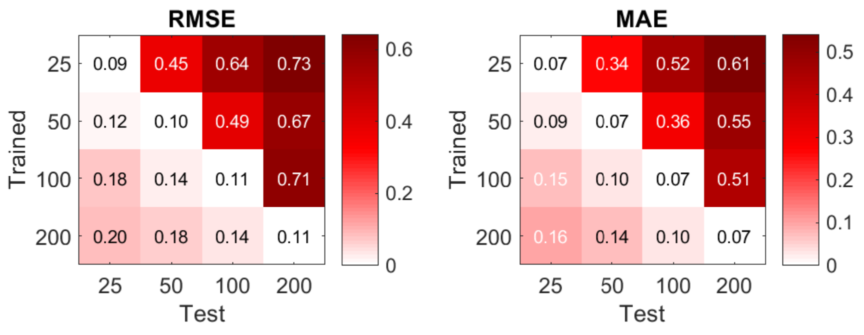


(c) Laser

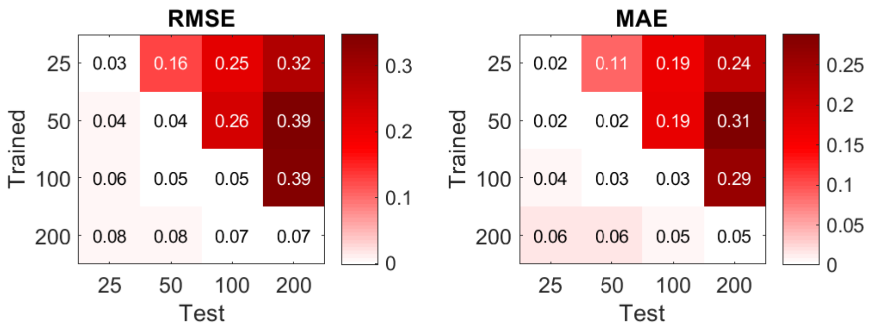


(d) Non-linear Channel Equalisation

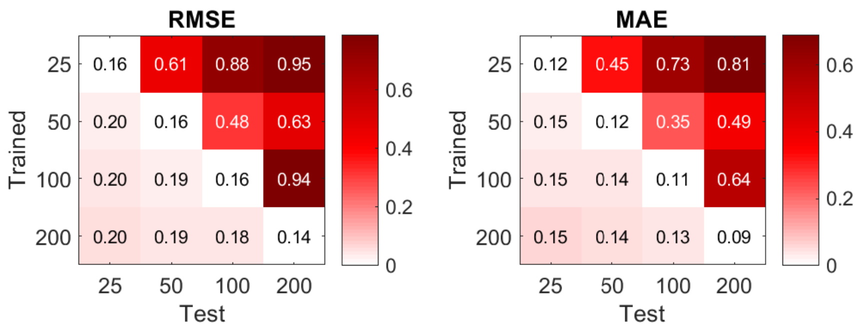
Figure E.5: FFNN: Features=3, Threshold=1.



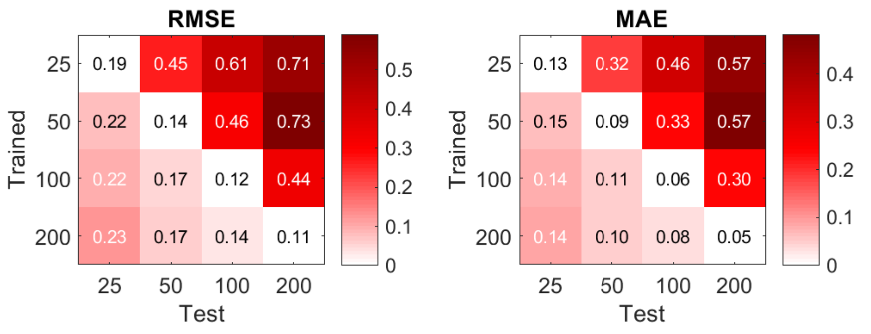
(a) NARMA-10



(b) NARMA-30

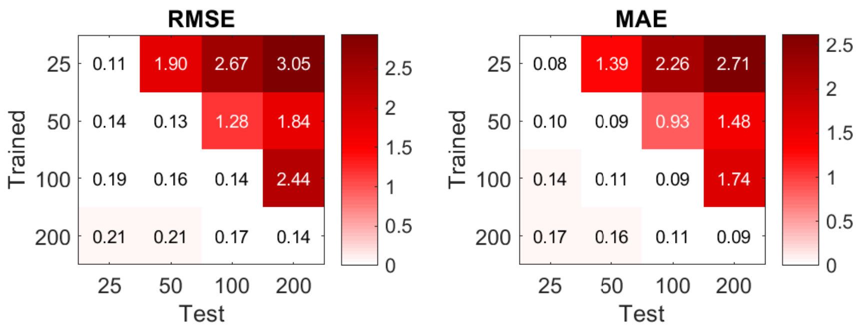


(c) Laser

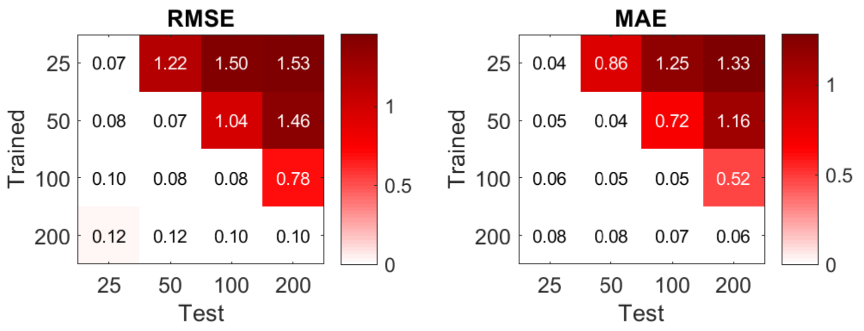


(d) Non-linear Channel Equalisation

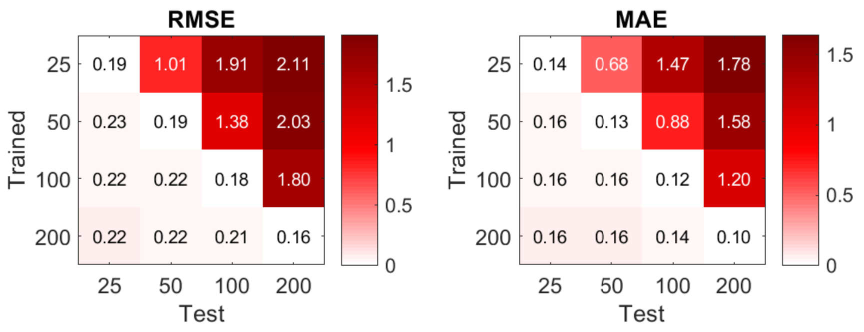
Figure E.6: FFNN: Features=3, Threshold=0.8.



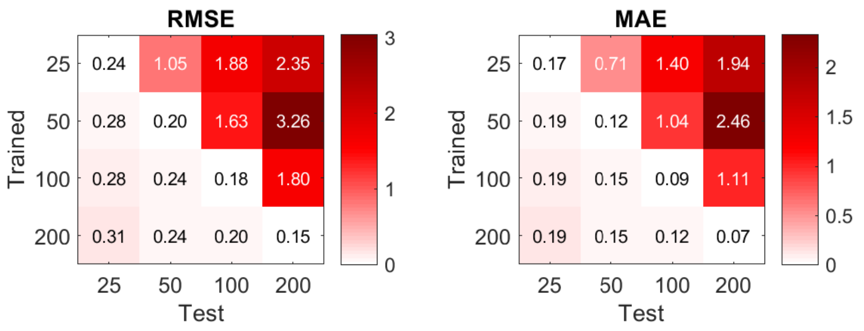
(a) NARMA-10



(b) NARMA-30

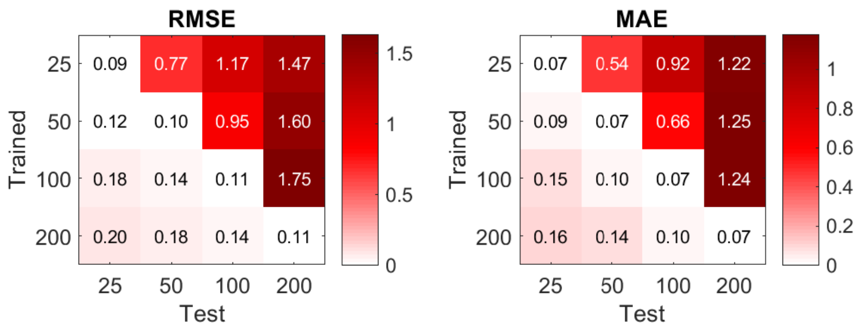


(c) Laser

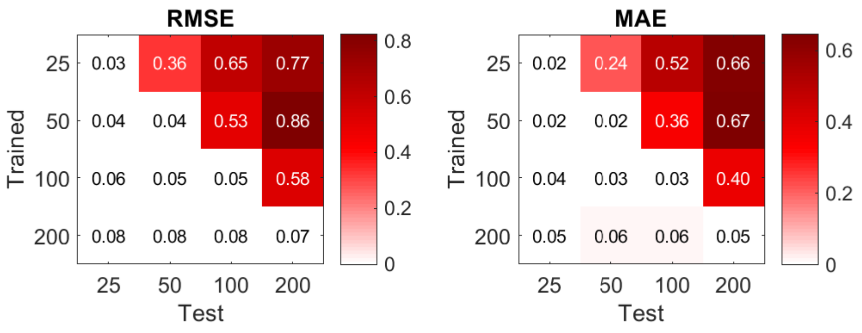


(d) Non-linear Channel Equalisation

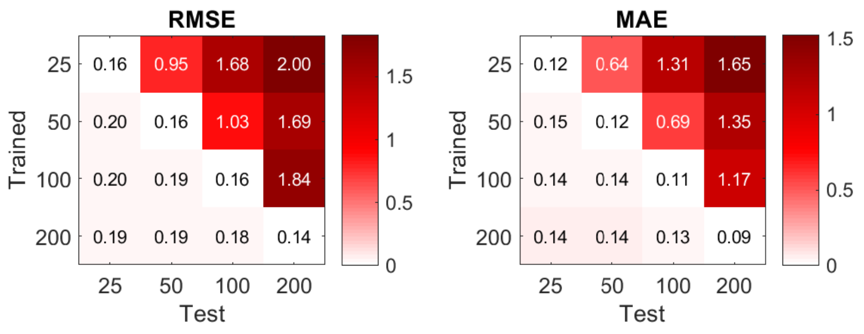
Figure E.7: FFNN: Features=11, Threshold=1.



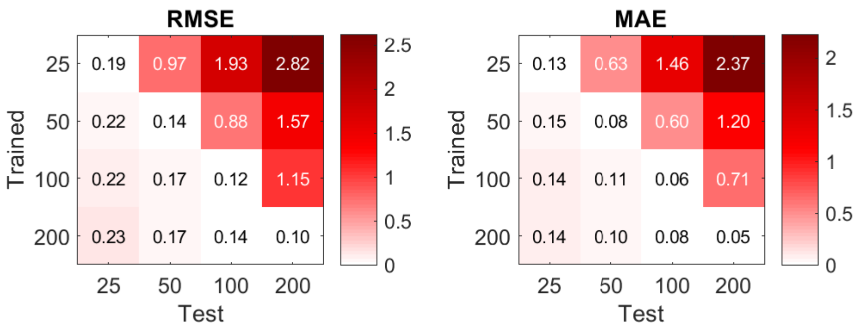
(a) NARMA-10



(b) NARMA-30

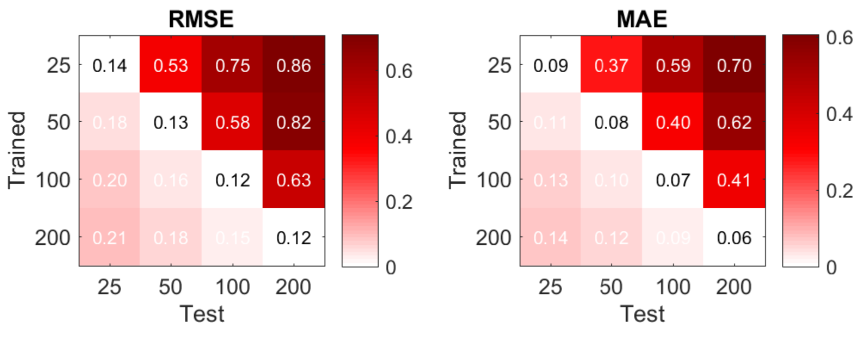


(c) Laser

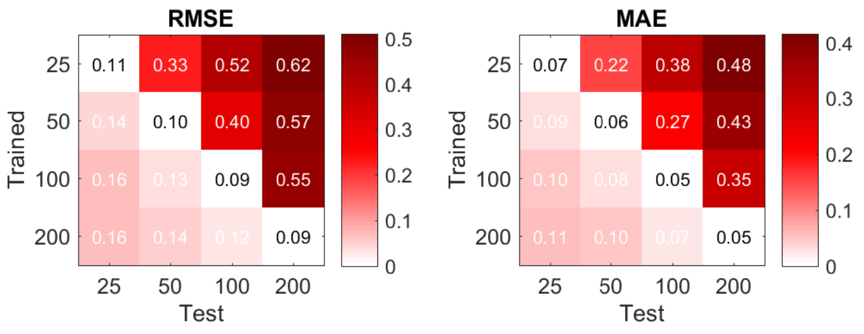


(d) Non-linear Channel Equalisation

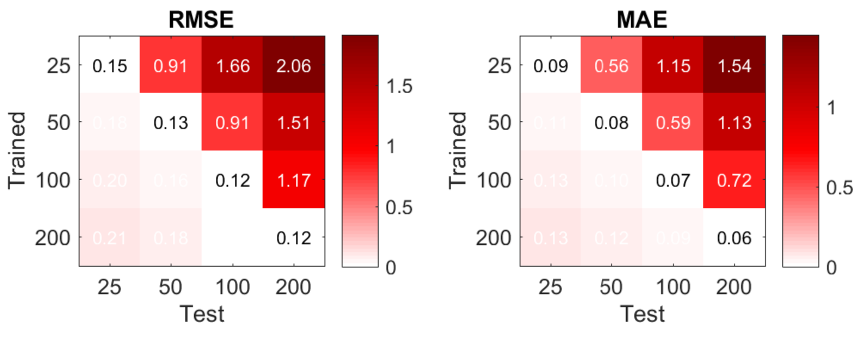
Figure E.8: FFNN: Features=11, Threshold=0.8.



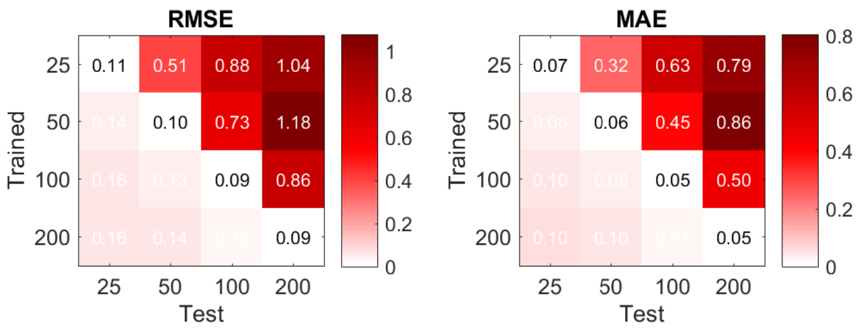
(a) Features=3, Threshold=1



(b) Features=3, Threshold=0.8

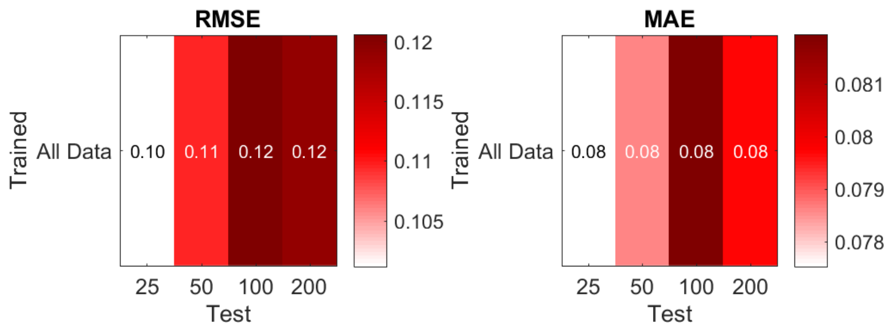


(c) Features=11, Threshold=1

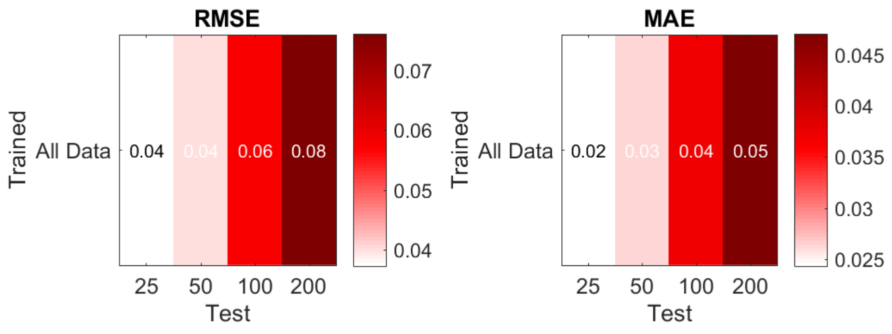


(d) Features=11, Threshold=0.8

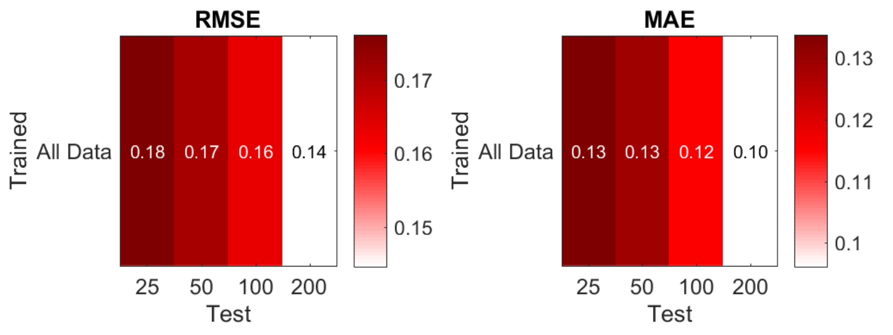
Figure E.9: Multi-Task FFNNs.



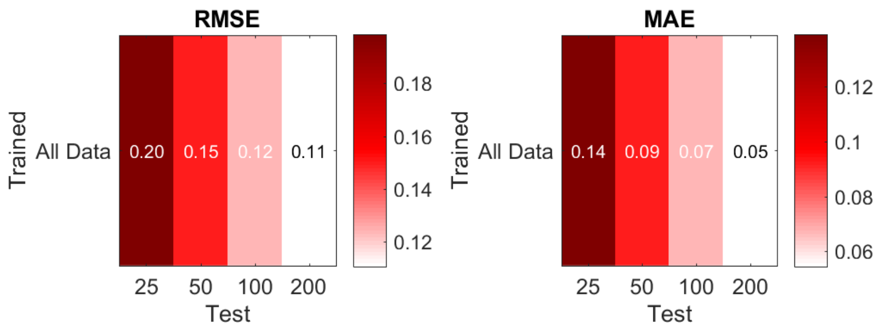
(a) NARMA-10



(b) NARMA-30



(c) Laser



(d) Non-linear Channel Equalisation

Figure E.10: FFNN trained on all ESN sizes. Features = 3, Threshold = 0.8.

Bibliography

[1] A. Adamatzky, editor. *Advances in Unconventional Computing: Volume 1: Theory*. Springer, 2016.

[2] A. Adamatzky, editor. *Advances in Unconventional Computing: Volume 2 Prototypes, Models and Algorithms*. Springer, 2016.

[3] A. Adamatzky, L. Bull, B. D. L. Costello, S. Stepney, and C. Teuscher, editors. *Unconventional Computing 2007*. Luniver Press, 2007.

[4] A. Adamatzky, B. Costello, and T. Asai. *Reaction-diffusion computers*. Elsevier, 2005.

[5] A. Adamatzky, V. Erokhin, M. Grube, T. Schubert, and A. Schumann. Physarum chip project: Growing computers from slime mould. *IJUC*, 8(4):319–323, 2012.

[6] S. G. Akl. The myth of universal computation. In M. Vajteršic, R. Trobec, P. Zinterhof, and A. Uhl, editors, *Parallel Numerics ’05*, pages 167–192, 2005.

[7] S. G. Akl. Nonuniversality in computation: Fifteen misconceptions rectified. In A. Adamatzky, editor, *Advances in Unconventional Computing: Volume 1: Theory*, pages 1–30. Springer, 2017.

[8] E. A. Antonelo, B. Schrauwen, and J. Van Campenhout. Generative modeling of autonomous robots and their environments using reservoir computing. *Neural Processing Letters*, 26(3):233–249, 2007.

[9] L. Appeltant. Reservoir computing based on delay-dynamical systems. *These de Doctorat, Vrije Universiteit Brussel/Universitat de les Illes Balears*, 2012.

[10] L. Appeltant, M. C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C. R. Mirasso, and I. Fischer. Information processing using a single dynamical node as complex system. *Nature Communications*, 2:468, 2011.

[11] A. F. Atiya and A. G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

[12] S. Basterrech, E. Alba, and V. Snášel. An experimental analysis of the echo state network initialization using the particle swarm optimization. In *Nature and Biologically Inspired Computing (NaBIC), 2014 Sixth World Congress on*, pages 214–219. IEEE, 2014.

[13] J. M. Beggs. The criticality hypothesis: how local cortical networks might optimize information processing. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 366(1864):329–343, 2008.

[14] A. Belkin, A. Hubler, and A. Bezryadin. Self-assembled wiggling nano-structures and the principle of maximum entropy production. *Scientific Reports*, 5, 2015. Article No. 8323.

[15] N. Bertschinger and T. Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.

[16] J. Boedecker, O. Obst, J. T. Lizier, N. M. Mayer, and M. Asada. Information processing in echo state networks at the edge of chaos. *Theory in Biosciences*, 131(3):205–213, 2012.

[17] S. Bose, C. Lawrence, Z. Liu, K. Makarenko, R. van Damme, H. Broersma, and W. van der Wiel. Evolution of a designless nanoparticle network into reconfigurable boolean logic. *Nature nanotechnology*, doi:10.1038/nnano.2015.207, 2015.

[18] H. Broersma, F. Gomez, J. Miller, M. Petty, and G. Tufte. Nascence project: Nanoscale engineering for novel computation using evolution. *International Journal of Unconventional Computing*, 8(4):313–317, 2012.

[19] H. Broersma, J. F. Miller, and S. Nichele. Computational matter: Evolving computational functions in nanoscale materials. In *Advances in Unconventional Computing*, pages 397–428. Springer, 2017.

[20] D. Brunner, M. C. Soriano, C. R. Mirasso, and I. Fischer. Parallel photonic information processing at gigabyte per second data rates using transient states. *Nature communications*, 4:1364, 2013.

[21] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Composing a reservoir of memristive networks for real-time computing. *arXiv preprint arXiv:1504.02833*, 2015.

[22] J. Bürger, A. Goudarzi, D. Stefanovic, and C. Teuscher. Hierarchical composition of memristive networks for real-time computing. In *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 33–38. IEEE, 2015.

[23] E. Burke, S. Gustafson, and G. Kendall. A survey and analysis of diversity measures in genetic programming. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 716–723. Morgan Kaufmann Publishers Inc., 2002.

[24] V. Bush. The differential analyzer. a new machine for solving differential equations. *Journal of the Franklin Institute*, 212(4):447–488, 1931.

[25] L. Büsing, B. Schrauwen, and R. Legenstein. Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. *Neural Computation*, 22(5):1272–1311, 2010.

[26] J. Butcher, D. Verstraeten, B. Schrauwen, C. Day, and P. Haycock. Extending reservoir computing with random static projections: a hybrid between extreme learning and rc. In *18th European Symposium on Artificial Neural Networks (ESANN 2010)*, pages 303–308. D-Side, 2010.

[27] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. Day, and P. Haycock. Reservoir computing and extreme learning machines for non-linear time-series data analysis. *Neural networks*, 38:76–89, 2013.

[28] K. Caluwaerts, M. D'Haene, D. Verstraeten, and B. Schrauwen. Locomotion without a brain: physical reservoir computing in tensegrity structures. *Artificial Life*, 19(1):35–66, 2013.

[29] J. P. Carbajal, J. Dambre, M. Hermans, and B. Schrauwen. Memristor models for machine learning. *Neural Computation*, 27(3):725–747, 2015.

[30] P. Cariani. To evolve an ear. epistemological implications of Gordon Pask's electrochemical devices. *Systems research*, 10(3):19–33, 1993.

[31] K. C. Chatzidimitriou and P. A. Mitkas. A neat way for evolving echo state networks. In *Proceedings of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 909–914. IOS Press, 2010.

[32] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[33] J. Chrol-Cannon and Y. Jin. On the correlation between reservoir metrics and performance for time series classification under the influence of synaptic plasticity. *PloS one*, 9(7):e101792, 2014.

[34] K. D. Clegg, J. F. Miller, M. K. Massey, and M. Petty. Travelling salesman problem solved 'in materio' by evolved carbon nanotube device. In *Parallel Problem Solving from Nature–PPSN XIII*, pages 692–701. Springer, 2014.

[35] K. D. Clegg, J. F. Miller, M. K. Massey, and M. C. Petty. Practical issues for configuring carbon nanotube composite materials for computation. In *ICES 2014, IEEE International Conference on Evolvable Systems*, pages 61–68. IEEE, 2014.

[36] M. Conrad. The price of programmability. *The universal turing machine a half-century survey*, pages 261–281, 1995.

[37] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[38] M. Dale. Neuroevolution of hierarchical reservoir computers. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 410–417. ACM, 2018.

[39] M. Dale, J. F. Miller, and S. Stepney. Reservoir computing as a model for in-materio computing. In *Advances in Unconventional Computing*, pages 533–571. Springer, 2017.

[40] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer. Evolving carbon nanotube reservoir computers. In *International Conference on Unconventional Computation and Natural Computation*, pages 49–61. Springer, 2016.

[41] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer. Reservoir computing in materio: An evaluation of configuration through evolution. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016.

[42] M. Dale, J. F. Miller, S. Stepney, and M. A. Trefzer. Reservoir computing in materio: A computational framework for in materio computing. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2178–2185, May 2017.

[43] J. Dambre, D. Verstraeten, B. Schrauwen, and S. Massar. Information processing capacity of dynamical systems. *Scientific Reports*, 2, 2012.

[44] S. Dasgupta, F. Wörgötter, and P. Manoonpong. Information theoretic self-organised adaptation in reservoirs for temporal memory tasks. In *Engineering Applications of Neural Networks*, pages 31–40. Springer, 2012.

[45] E. D. De Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pages 11–18. Morgan Kaufmann Publishers Inc., 2001.

[46] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, 2005.

[47] E. C. Demis, R. Aguilera, K. Scharnhorst, M. Aono, A. Z. Stieg, and J. K. Gimzewski. Nanoarchitectonic atomic switch networks for unconventional computing. *Japanese Journal of Applied Physics*, 55(11):1102B2, 2016.

[48] Z. Deng and Y. Zhang. Collective behavior of a small-world recurrent neural system with scale-free distribution. *IEEE Transactions on Neural Networks*, 18(5):1364–1375, 2007.

[49] B. Derrida and Y. Pomeau. Random networks of automata: a simple annealed approximation. *EPL (Europhysics Letters)*, 1(2):45, 1986.

[50] P. F. Dominey. Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274, 1995.

[51] C. Du, F. Cai, M. A. Zidan, W. Ma, S. H. Lee, and W. D. Lu. Reservoir computing using dynamic memristors for temporal information processing. *Nature communications*, 8(1):2204, 2017.

[52] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, and M. Nuttin. Pruning and regularization in reservoir computing. *Neurocomputing*, 72(7-9):1534–1546, 2009.

[53] A. Eiben, S. Kernbach, and E. Haasdijk. Embodied artificial evolution. *Evolutionary intelligence*, 5(4):261–272, 2012.

[54] M. J. Embrechts, L. A. Alexandre, and J. D. Linton. Reservoir computing for static pattern recognition. In *17th European Symposium on Artificial Neural Networks (ESANN 2009)*, 2009.

[55] C. Fernando and S. Sojakka. Pattern recognition in a bucket. In *Advances in Artificial Life*, pages 588–597. Springer, 2003.

[56] A. A. Ferreira and T. B. Ludermir. Comparing evolutionary methods for reservoir computing pre-training. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 283–290. IEEE, 2011.

[57] A. A. Ferreira, T. B. Ludermir, and R. R. De Aquino. An approach to reservoir computing design and training. *Expert systems with applications*, 40(10):4172–4182, 2013.

[58] M. Fiers, T. Van Vaerenbergh, F. Wyffels, D. Verstraeten, D. J. Schrauwen, B., and P. Bienstman. Nanophotonic reservoir computing with photonic crystal cavities to generate periodic patterns. *IEEE Transactions on Neural Networks and Learning Systems*, 25(2):344–355, 2014.

[59] K. Fujii and K. Nakajima. Harnessing disordered-ensemble quantum dynamics for machine learning. *Physical Review Applied*, 8(2):024030, 2017.

[60] K. Funahashi and Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806, 1993.

[61] C. Gallicchio and A. Micheli. Architectural and markovian factors of echo state networks. *Neural Networks*, 24(5):440–456, 2011.

[62] C. Gallicchio, A. Micheli, and L. Pedrelli. Deep reservoir computing: a critical experimental analysis. *Neurocomputing*, 2017.

[63] T. E. Gibbons. Unifying quality metrics for reservoir networks. In *IJCNN 2010, The International Joint Conference on Neural Networks*, pages 1–7. IEEE, 2010.

[64] D. E. Goldberg, J. Richardson, et al. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[65] A. Goudarzi and C. Teuscher. Reservoir computing: Quo vadis? In *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication*, page 13. ACM, 2016.

[66] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[67] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[68] G. W. Greenwood and A. M. Tyrrell. *Introduction to evolvable hardware: a practical guide for designing self-adaptive systems*, volume 5. John Wiley & Sons, 2006.

[69] K. Greff, R. Damme, J. Koutnik, H. Broersma, J. Mikhal, C. Lawrence, W. Wiel, and J. Schmidhuber. Unconventional computing using evolution-in-nanomaterio: neural networks meet nanoparticle networks. *Eighth International Conference on Future Computational Technologies and Applications, FUTURE COMPUTING 2016*, pages 15–20, 2016.

[70] J. M. Gutierrez, T. Hinkley, J. W. Taylor, K. Yanev, and L. Cronin. Evolution of oil droplets in a chemorobotic platform. *Nature Communications*, 5, 2014.

[71] P. C. Haddow and A. M. Tyrrell. Challenges of evolvable hardware: past, present and the path to a promising future. *Genetic Programming and Evolvable Machines*, 12(3):183–215, 2011.

[72] S. Harding. *Evolution in materio*. PhD thesis, University of York, 2005.

[73] S. Harding and J. F. Miller. Evolution in materio: Initial experiments with liquid crystal. In *2004 NASA/DoD Conference on Evolvable Hardware*, pages 298–305. IEEE, 2004.

[74] S. Harding and J. F. Miller. Evolution in materio: A real-time robot controller in liquid crystal. In *2005 NASA/DoD Conference on Evolvable Hardware*, pages 229–238. IEEE, 2005.

[75] S. Harding and J. F. Miller. Evolution in materio: Evolving logic gates in liquid crystal. In *ECAL 2005 Workshop on Unconventional Computing: From cellular automata to wetware*, pages 133–149. Beckington, UK, 2005.

[76] I. Harvey. The microbial genetic algorithm. In *European Conference on Artificial Life*, pages 126–133. Springer, 2009.

[77] M. Hénon. A two-dimensional mapping with a strange attractor. *Communications in Mathematical Physics*, 50(1):69–77, 1976.

[78] K. Hicke, M. A. Escalona-Morán, D. Brunner, M. C. Soriano, I. Fischer, and C. R. Mirasso. Information processing using transient dynamics of semiconductor lasers subject to delayed feedback. *IEEE Journal of Selected Topics in Quantum Electronics*, 19(4):1501610–1501610, 2013.

[79] T. Higuchi, M. Iwata, I. Kajitani, H. Yamada, B. Manderick, Y. Hirao, M. Murakawa, S. Yoshizawa, and T. Furuya. Evolvable hardware with genetic learning. In *ISCAS'96, IEEE International Symposium on Circuits and Systems*, volume 4, pages 29–32. IEEE, 1996.

[80] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[81] C. Horsman, S. Stepney, R. C. Wagner, and V. Kendon. When does a physical system compute? *Proc. R. Soc. A*, 470(2169):20140182, 2014.

[82] D. C. Horsman. Abstraction/representation theory for heterotic physical computing. *Phil. Trans. R. Soc. A*, 373(2046):20140224, 2015.

[83] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[84] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[85] M. Inubushi and K. Yoshimura. Reservoir computing beyond memory-nonlinearity trade-off. *Scientific Reports*, 7(1):10199, 2017.

[86] H. Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.

[87] H. Jaeger. *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik, 2001.

[88] H. Jaeger. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pages 593–600, 2002.

[89] H. Jaeger. Discovering multiscale dynamical features with hierarchical echo state networks. *Technical report No. 9*, 2007.

[90] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[91] H. Jaeger, M. Lukoševičius, D. Popovici, and U. Siewert. Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352, 2007.

[92] S. Jarvis, S. Rotter, and U. Egert. Extending stability through hierarchical clusters in echo state networks. *Frontiers in neuroinformatics*, 4, 2010.

[93] F. Jiang, H. Berry, and M. Schoenauer. Supervised and evolutionary learning of echo state networks. In *Parallel Problem Solving from Nature–PPSN X*, pages 215–224. Springer, 2008.

[94] B. Jones, D. Stekel, J. Rowe, and C. Fernando. Is there a liquid state machine in the bacterium escherichia coli? In *Artificial Life, 2007. ALIFE’07. IEEE Symposium on*, pages 187–191. IEEE, 2007.

[95] S. Kauffman. Homeostasis and differentiation in random genetic control networks. *Nature*, 224(5215):177, 1969.

[96] V. Kendon, A. Sebald, S. Stepney, M. Bechmann, P. Hines, and R. C. Wagner. Heterotic computing. In *International Conference on Unconventional Computation*, pages 113–124. Springer, 2011.

[97] J. Kilian and H. T. Siegelmann. The dynamic universality of sigmoidal neural networks. *Information and computation*, 128(1):48–56, 1996.

[98] S. Klampfl, S. David, P. Yin, S. Shamma, and W. Maass. Integration of stimulus history in information conveyed by neurons in primary auditory cortex in response to tone sequences. In *39th Annual Conference of the Society for Neuroscience, Program*, volume 163, 2009.

[99] Z. Konkoli and G. Wendin. On information processing with networks of nano-scale switching elements. *International Journal of Unconventional Computing*, 10(5-6):405–428, 2014.

[100] A. Kotsialos, M. Massey, F. Qaiser, D. Zeze, C. Pearson, and M. Petty. Logic gate and circuit training on randomly dispersed carbon nanotubes. *International journal of unconventional computing.*, 10(5-6):473–497, 2014.

[101] A. U. Küçükemre. *Echo state networks for adaptive filtering*. PhD thesis, University of Applied Sciences, 2006.

[102] M. S. Kulkarni and C. Teuscher. Memristor-based reservoir computing. In *NANOARCH, 2012, IEEE/ACM International Symposium on Nanoscale Architectures*, pages 226–232. IEEE, 2012.

[103] C. G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Physica D: Nonlinear Phenomena*, 42(1):12–37, 1990.

[104] L. Larger, M. C. Soriano, D. Brunner, L. Appeltant, J. M. Gutiérrez, L. Pesquera, C. R. Mirasso, and I. Fischer. Photonic information processing beyond turing: an optoelectronic implementation of reservoir computing. *Optics Express*, 20(3):3241–3249, 2012.

[105] R. Legenstein and W. Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007.

[106] R. Legenstein and W. Maass. What makes a dynamical system computationally powerful. *New directions in statistical signal processing: From systems to brain*, pages 127–154, 2007.

[107] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[108] J. Lehman and K. O. Stanley. Efficiently evolving programs through the search for novelty. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 837–844. ACM, 2010.

[109] J. Lehman and K. O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 103–110. ACM, 2010.

[110] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[111] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406(6799):974–978, 2000.

[112] J. T. Lizier. Jidt: An information-theoretic toolkit for studying the dynamics of complex systems. *Frontiers in Robotics and AI*, 1:11, 2014.

[113] J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Local information transfer as a spatiotemporal filter for complex systems. *Physical Review E*, 77(2):026110, 2008.

[114] J. T. Lizier, M. Prokopenko, and A. Y. Zomaya. Local measures of information storage in complex distributed computation. *Information Sciences*, 208:39–54, 2012.

[115] S. Lloyd. Ultimate physical limits to computation. *Nature*, 406(6799):1047, 2000.

[116] J. D. Lohn, D. S. Linden, G. S. Hornby, W. F. Kraus, and A. Rodriguez-Arroyo. Evolutionary design of an X-band antenna for NASA's space technology 5 mission. In *NASA/DoD Conference on Evolvable Hardware*, pages 155–155. IEEE, 2003.

[117] M. Lukoševičius. A practical guide to applying echo state networks. In *Neural Networks: Tricks of the Trade*, pages 659–686. Springer, 2012.

[118] M. Lukoševičius and H. Jaeger. Overview of reservoir recipes. Technical Report 11, Jacobs University Bremen, 2007.

[119] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[120] M. Lukoševičius, H. Jaeger, and B. Schrauwen. Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.

[121] M. Lukoševičius, D. Popovici, H. Jaeger, and U. Siewert. Time warping invariant echo state networks. Technical report, International University of Bremen, School of Engineering and Science, 2006.

[122] O. R. Lykkebø, S. Harding, G. Tufte, and J. F. Miller. Mecobo: A hardware and software platform for in materio evolution. In *Unconventional Computation and Natural Computation*, pages 267–279. Springer, 2014.

[123] O. R. Lykkebø and G. Tufte. Comparison and evaluation of signal representations for a carbon nanotube computational device. In *IEEE International Conference on Evolvable Systems (ICES 2014)*, pages 54–60. IEEE, 2014.

[124] Q. Ma, L. Shen, and G. W. Cottrell. Deep-esn: A multiple projection-encoding hierarchical reservoir computing framework. *arXiv preprint arXiv:1711.05255*, 2017.

[125] Q. Ma, L. Shen, W. Zhuang, and J. Chen. Decouple adversarial capacities with dual-reservoir network. In *International Conference on Neural Information Processing*, pages 475–483. Springer, 2017.

[126] W. Maass. Liquid state machines: motivation, theory, and applications. *Computability in context: computation and logic in the real world*, pages 275–296, 2010.

[127] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[128] K. Machines. Indiana university computer science technical report 439, 1995.

[129] B. J. MacLennan. A review of analog computing. *Department of Electrical Engineering & Computer Science, University of Tennessee, Technical Report UT-CS-07-601 (September)*, 2007.

[130] M. Massey, A. Kotsialos, F. Qaiser, D. Zeze, C. Pearson, D. Volpati, L. Bowen, and M. Petty. Computing with carbon nanotubes: Optimization of threshold logic gates using disordered nanotube/polymer composites. *Journal of Applied Physics*, 117(13):134903, 2015.

[131] M. Massey, A. Kotsialos, D. Volpati, E. Vissol-Gaudin, C. Pearson, L. Bowen, B. Obara, D. Zeze, C. Groves, and M. Petty. Evolution of electronic circuits using carbon nanotube composites. *Scientific Reports*, 6, 2016.

[132] M. K. Massey, C. Pearson, D. A. Zeze, B. G. Mendis, and M. C. Petty. The electrical and optical properties of oriented langmuir-blodgett films of single-walled carbon nanotubes. *Carbon*, 49(7):2424–2430, 2011.

[133] F. Matzner. Neuroevolution on the edge of chaos. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 465–472. ACM, 2017.

[134] J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In *NASA/DoD Conference on Evolvable Hardware 2002*, pages 167–176. IEEE, 2002.

[135] J. F. Miller, S. Harding, and G. Tufte. Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7(1):49–67, 2014.

[136] J. W. Mills. Polymer processors. Technical report, Technical Report TR580, Department of Computer Science, University of Indiana, 1995.

[137] J. W. Mills. The nature of the extended analog computer. *Physica D: Nonlinear Phenomena*, 237(9):1235–1256, 2008.

[138] M. Mohid and J. F. Miller. Evolving robot controllers using carbon nanotubes. In *13th European Conference on Artificial Life*, 2015.

[139] M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebø, M. K. Massey, and M. C. Petty. Evolution-in-materio: A frequency classifier using materials. In *ICES 2014, International Conference on Evolvable Systems*, pages 46–53. IEEE, 2014.

[140] M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebø, M. K. Massey, and M. C. Petty. Evolution-in-materio: Solving bin packing problems using materials. In *ICES 2014, International Conference on Evolvable Systems*, pages 38–45. IEEE, 2014.

[141] M. Mohid, J. F. Miller, S. Harding, G. Tufte, O. R. Lykkebø, M. K. Massey, and M. C. Petty. Evolution-in-materio: Solving machine learning classification problems using materials. In *PPSN XIII, Parallel Problem Solving from Nature*, pages 721–730. Springer, 2014.

[142] M. Mohid, J. F. Miller, S. L. Harding, G. Tufte, M. K. Massey, and M. C. Petty. Evolving solutions to computational problems using carbon nanotubes. *International Journal of Unconventional Computing*, 11(3/4):245–281, 2015.

[143] M. Mohid, J. F. Miller, S. L. Harding, G. Tufte, M. K. Massey, and M. C. Petty. Evolution-in-materio: solving computational problems using carbon nanotube–polymer composites. *Soft Computing*, 20(8):3007–3022, 2016.

[144] K. Nakajima, H. Hauser, R. Kang, E. Guglielmino, D. G. Caldwell, and R. Pfeifer. A soft body as a reservoir: case studies in a dynamic model of octopus-inspired soft robotic arm. *Frontiers in computational neuroscience*, 7:91, 2013.

[145] R. M. Nguimdo, G. Verschaffelt, J. Danckaert, and G. Van der Sande. Reducing the phase sensitivity of laser-based optical reservoir computing systems. *Optics express*, 24(2):1238–1252, 2016.

[146] S. Nichele, D. Laketic, O. R. Lykkebø, and G. Tufte. Is there chaos in blobs of carbon nanotubes used to perform computation? In *Proceedings of 7th International Conference on Future Comp. Tech. and Applications. XPS Press*, pages 12–17, 2015.

[147] S. Nichele, O. R. Lykkebø, and G. Tufte. An investigation of underlying physical properties exploited by evolution in nanotubes materials. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 1220–1228. IEEE, 2015.

[148] D. Nikolić, S. Haeusler, W. Singer, and W. Maass. Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in neural information processing systems*, pages 1041–1048, 2006.

[149] C. R. Noback, N. L. Strominger, R. J. Demarest, and D. A. Ruggiero. *The human nervous system: structure and function*. Number 744. Springer Science & Business Media, 2005.

[150] D. Norton and D. Ventura. Improving liquid state machines through iterative refinement of the reservoir. *Neurocomputing*, 73(16):2893–2904, 2010.

[151] O. Obst, A. Trinchi, S. G. Hardin, M. Chadwick, I. Cole, T. H. Muster, N. Hoschke, D. Ostry, D. Price, K. N. Pham, et al. Nano-scale reservoir computing. *Nano Communication Networks*, 4(4):189–196, 2013.

[152] N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, 1988.

[153] Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, and S. Massar. Optoelectronic reservoir computing. *Scientific Reports*, 2, 2012.

[154] G. Pask. Physical analogues to the growth of a concept. In *Mechanization of Thought Processes, Symposium*, volume 10, pages 765–794, 1958.

[155] D. Prychynenko, M. Sitte, K. Litzius, B. Krüger, G. Bourianoff, M. Kläui, J. Sinova, and K. Everschor-Sitte. Magnetic skyrmion as a nonlinear resistive element: A potential building block for reservoir computing. *Physical Review Applied*, 9(1):014034, 2018.

[156] J. Qiao, F. Li, H. Han, and W. Li. Growing echo-state network with multiple subreservoirs. *IEEE transactions on neural networks and learning systems*, 28(2):391–404, 2017.

[157] S. Risi, S. D. Vanderbleek, C. E. Hughes, and K. O. Stanley. How novelty search escapes the deceptive trap of learning to learn. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 153–160. ACM, 2009.

[158] A. Rodan and P. Tino. Minimum complexity echo state network. *IEEE Transactions on Neural Networks*, 22(1):131–144, 2011.

[159] L. A. Rubel. The extended analog computer. *Advances in Applied Mathematics*, 14(1):39–50, 1993.

[160] E. Samuelsen and K. Glette. Real-world reproduction of evolved robot morphologies: Automated categorization and evaluation. In *Applications of Evolutionary Computation*, volume 9028 of *LNCS*, pages 771–782. Springer, 2015.

[161] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural computation*, 19(3):757–779, 2007.

[162] B. Scholkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[163] B. Schrauwen, J. Defour, D. Verstraeten, and J. Van Campenhout. The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Artificial Neural Networks–ICANN 2007*, pages 471–479. Springer, 2007.

[164] B. Schrauwen, M. Wardermann, D. Verstraeten, J. J. Steil, and D. Stroobandt. Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7):1159–1171, 2008.

[165] T. Schreiber. Measuring information transfer. *Physical review letters*, 85(2):461, 2000.

[166] F. Schwenker and A. Labib. Echo state networks and neural network ensembles to predict sunspots activity. In *proceedings European Symposium on Artificial Neural Networks-Advances in Computational Intelligence and Learning (ESANN), Bruges*. Citeseer, 2009.

[167] A. T. Sergio and T. B. Ludermir. Pso for reservoir computing optimization. In *International Conference on Artificial Neural Networks*, pages 685–692. Springer, 2012.

[168] C. E. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337–354, 1941.

[169] C. E. Shannon and W. Weaver. The mathematical theory of communication. *Urbana: University of Illinois Press*, 1949.

[170] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[171] H. O. Sillin, R. Aguilera, H. Shieh, A. V. Avizienis, M. Aono, A. Z. Stieg, and J. K. Gimzewski. A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing. *Nanotechnology*, 24(38):384004, 2013.

[172] N. Soures, C. Merkel, D. Kudithipudi, C. Thiem, and N. McDonald. Reservoir computing in embedded systems: Three variants of the reservoir algorithm. *IEEE Consumer Electronics Magazine*, 6(3):67–73, 2017.

[173] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[174] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[175] A. Steane. Quantum computing. *Reports on Progress in Physics*, 61(2):117, 1998.

[176] J. J. Steil. Backpropagation-decorrelation: online recurrent learning with o (n) complexity. In *2004 IEEE International Joint Conference on Neural Networks*, volume 2, pages 843–848. IEEE, 2004.

[177] J. J. Steil. Online reservoir adaptation by intrinsic plasticity for backpropagation–decorrelation and echo state learning. *Neural Networks*, 20(3):353–364, 2007.

[178] S. Stepney. The neglected pillar of material computation. *Physica D: Nonlinear Phenomena*, 237(9):1157–1164, 2008.

[179] S. Stepney. Nonclassical computation: a dynamical systems perspective. In G. Rozenberg, T. Bäck, and J. N. Kok, editors, *Handbook of Natural Computing, volume 4*, pages 1979–2025. Springer, 2012.

[180] S. Stepney, S. Abramsky, A. Adamatzky, C. Johnson, and J. Timmis. Grand challenge 7: Journeys in non-classical computation. In *Visions of Computer Science, London, UK, September 2008*, pages 407–421. BCS, 2008.

[181] S. Stepney, S. L. Braunstein, J. A. Clark, A. Tyrrell, A. Adamatzky, R. E. Smith, T. Addis, C. Johnson, J. Timmis, and P. Welch. Journeys in non-classical computation I: A grand challenge for computing research. *International Journal of Parallel, Emergent and Distributed Systems*, 20(1):5–19, 2005.

[182] S. Stepney, V. Kendon, P. Hines, and A. Sebald. A framework for heterotic computing. *arXiv preprint arXiv:1210.0621*, 2012.

[183] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, R. Aguilera, H. Shieh, C. Martin-Olmos, E. J. Sandouk, M. Aono, and J. K. Gimzewski. Self-organization and emergence of dynamical structures in neuromorphic atomic switch networks. In *Memristor Networks*, pages 173–209. Springer, 2014.

[184] A. Z. Stieg, A. V. Avizienis, H. O. Sillin, C. Martin-Olmos, M. Aono, and J. K. Gimzewski. Emergent criticality in complex Turing B-type atomic switch networks. *Advanced Materials*, 24(2):286–293, 2012.

[185] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose. Recent advances in physical reservoir computing: A review. *arXiv preprint arXiv:1808.04962*, 2018.

[186] H. Tanaka, M. Akai-Kasaya, A. TermehYousefi, L. Hong, L. Fu, H. Tamukoh, D. Tanaka, T. Asai, and T. Ogawa. A molecular neuromorphic network device consisting of single-walled carbon nanotubes complexed with polyoxometalate. *Nature communications*, 9, 2018.

[187] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *Evolvable Systems: From Biology to Hardware*, pages 390–405. Springer, 1997.

[188] A. Toffolo and E. Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary computation*, 11(2):151–167, 2003.

[189] J. Torrejon, M. Riou, F. A. Araujo, S. Tsunegi, G. Khalsa, D. Querlioz, P. Bortolotti, V. Cros, K. Yakushiji, A. Fukushima, et al. Neuromorphic computing with nanoscale spintronic oscillators. *Nature*, 547(7664):428, 2017.

[190] S. D. Tran and C. Teuscher. Memcapacitive reservoir computing. In *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 115–116. IEEE, 2017.

[191] K. Vandoorne, J. Dambre, D. Verstraeten, B. Schrauwen, and P. Bienstman. Parallel reservoir computing using optical amplifiers. *Neural Networks, IEEE Transactions on*, 22(9):1469–1481, 2011.

[192] K. Vandoorne, P. Mechet, T. Van Vaerenbergh, M. Fiers, G. Morthier, D. Verstraeten, B. Schrauwen, J. Dambre, and P. Bienstman. Experimental demonstration of reservoir computing on a silicon photonics chip. *Nature Communications*, 5:3541, 2014.

[193] A. Vargha and H. D. Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[194] D. Verstraeten, J. Dambre, X. Dutoit, and B. Schrauwen. Memory versus nonlinearity in reservoirs. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2010.

[195] D. Verstraeten and B. Schrauwen. On the quantification of dynamics in reservoir computing. In *Artificial Neural Networks–ICANN 2009*, pages 985–994. Springer, 2009.

[196] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007.

[197] Y. Viero, D. Guérin, A. Vladyka, F. Alibart, S. Lenfant, M. Calame, and D. Vuillaume. Light-stimulatable molecules/nanoparticles networks for switchable logical functions and reservoir computing. *Advanced Functional Materials*, page 1801506, 2018.

[198] E. Vissol-Gaudin, A. Kotsialos, M. K. Massey, D. A. Zeze, C. Pearson, C. Groves, and M. C. Petty. Training a carbon-nanotube/liquid crystal data classifier using evolutionary algorithms. In *International Conference on Unconventional Computation and Natural Computation*, pages 130–141. Springer, 2016.

[199] D. Volpati, M. Massey, D. Johnson, A. Kotsialos, F. Qaiser, C. Pearson, K. Coleman, G. Tiburzi, D. Zeze, and M. Petty. Exploring the alignment of carbon nanotubes dispersed in a liquid crystal matrix using coplanar electrodes. *Journal of Applied Physics*, 117(12):125303, 2015.

[200] A. Weigend. *The Santa Fe Time Series Competition Data: Data set A, Laser generated data*, 1991 (accessed March, 2016).

[201] J. Whiting, B. de Lacy Costello, and A. Adamatzky. Slime mould logic gates based on frequency changes of electrical potential oscillation. *Biosystems*, 124:21–25, 2014.

[202] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of modern physics*, 55(3):601, 1983.

[203] Y. Xue, L. Yang, and S. Haykin. Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20(3):365–376, 2007.

[204] J. Yperman and T. Becker. Bayesian optimization of hyper-parameters in reservoir computing. *arXiv preprint arXiv:1611.05193*, 2016.

[205] S. Yuenyong. On the gradient-based sequential tuning of the echo state network reservoir parameters. In *Pacific Rim International Conference on Artificial Intelligence*, pages 651–660. Springer, 2016.

[206] S. Yuenyong and A. Nishihara. Evolutionary pre-training for crj-type reservoir of echo state networks. *Neurocomputing*, 149:1324–1329, 2015.

[207] S. Zhong, X. Xie, L. Lin, and F. Wang. Genetic algorithm optimized double-reservoir echo state network for multi-regime time series prediction. *Neurocomputing*, 238:191–204, 2017.