Reservoir Computing in Material Substrates

Matthew Dale, Julian F. Miller, Susan Stepney, and Martin A. Trefzer

Abstract We overview Reservoir Computing (RC) with physical systems from an Unconventional Computing (UC) perspective. We discuss challenges present in both fields, including *encoding* and *representation*, or how to manipulate and read information; ways to search large and complex configuration spaces of physical systems; and what makes a "*good*" computing substrate.

1 Introduction

As of 2018, there are many "flavours" and interpretations of physical reservoir computing systems. A recent review (Tanaka et al. 2019) classifies and groups these systems according to reservoir types based on physical properties exploited, such as chemical, optical, and mechanical based. These systems can vary immensely in terms of architecture, dynamics, degrees of freedom, ease to manipulate, size, complexity, and internal timescales. However, a general framework for physical reservoir computing and unconventional computing is still missing. The wide variety of potential computing systems presents several challenges: how best to design physical systems, how to determine what computational tasks they are best suited to, and how to assess and compare across different architectures.

In this chapter, we discuss three important aspects for describing any physical [reservoir] computing system: representation and instantiation; manipulation and programming; and what makes a "good" computing substrate. No matter what future

M. Dale · S. Stepney (✉)
Department of Computer Science, University of York, York, UK
e-mail: susan.stepney@york.ac.uk

M. Dale
e-mail: matt.dale@york.ac.uk

J. F. Miller · M. A. Trefzer
Department of Electronic Engineering, University of York, York, UK
e-mail: julian.miller@york.ac.uk

M. A. Trefzer
e-mail: martin.trefzer@york.ac.uk

K. Nakajima and I. Fischer (eds.), *Reservoir Computing*, Natural Computing Series,
https://doi.org/10.1007/978-981-13-1687-6_7

systems emerge, these three aspects will always play a pivotal role in the design and application of any physical (and virtual) reservoir computing system.

Physical systems tend to have high degrees of freedom or ways to be configured. Whether each configuration induces a small or large change in dynamical behaviour is more important to the system's computing ability. A substrate that can compute many different problems will possess a high degree of dynamical freedom, with the ability to instantiate many "reservoirs" in a single substrate.

Here, we use the term *reservoir* to refer to a specific configuration of some non-linear system, substrate, or device that facilitates computation. A configuration is a set of values of the parameters that control and influence the behaviour of the system. For reservoirs, these parameter values are typically static: they are held constant during the operation of the reservoir. In a simulated recurrent neural network, parameters may be weights or global scaling parameters. In an optical delay-based system, parameters define the sample-and-hold procedure, the length of a fibre optic delay line, or injection currents that shape the non-linearity of the system. In other systems, parameters include cell update rules in a cellular automaton, the volume of the reactor in a chemical reaction system, and the strength of the coupling between neighbouring oscillators in a mechanical oscillator network.

A reservoir is thus an abstract *representation* of a physical or simulated system, which is realised through the instantiation or configuration of physical parameters.

To successfully compute with a reservoir, in general, requires a physical system with a number of dynamical properties that exist internally, which can be driven by an external input signal. These properties, either present in a "natural" state or in a configured state, govern the information processing capabilities and capacity of the system: how information is stored, transmitted, distributed, and processed.

There are three known basic properties for reservoir computing. The *echo state* property (Jaeger 2001a) represents a fading memory, an essential property for learning time-dependent relationships for the prediction of future states based on previous states. The *separation* property represents a system's ability to project the input space into the high-dimensional phase space of the system. The *approximation* property (Maass et al. 2002) is the reservoir's ability to generalise given similar, or noisy, input signals, that is, to converge to the same attractor in terms of dynamics. Other "basic" properties may exist that are at present unknown.

Dambre et al. (2012) show that all dynamical systems featuring such properties possess the same total normalised capacity to process information. They identify and explain the importance of the non-linearity and memory trade-off for computing tasks, as well as the detrimental effect noise has on the computational capacity of dynamical systems. These insights, demonstrated with three dynamical systems, validate the RC framework's application to a range of underlying dynamical systems.

What type of dynamical systems the RC framework can be applied to is fairly broad, covering both discrete and continuous systems. However, common patterns exist for all systems. In each, information is processed via an *intrinsic* function, the physical system dynamics that transforms and maps input signals into observable system states. The details of how this intrinsic function works are generally irrelevant to the reservoir's programming process. The substrate, and subsequent reservoir, is a

black box. The model harnesses potentially unknown physical processes, performing functions on macroscopic behaviours resulting from unseen microscopic interactions.

In terms of programming physical and unconventional computers, the configuration, input, and output methods encompass the unconventional "program" of the system. Unlike conventional programs, instructions induce and exploit the intrinsic dynamics and state of the system without explicitly telling the system how to do so, or what to do at the lowest physical level. These programs are typically "learned" rather than hand-crafted. For example, the training of RC systems can be partitioned into multiple stages. Substrate parameters may be selected randomly (typically within a constrained domain) or trained through some optimisation process (Abubakar et al. 2018). The readout stage is typically learnt through linear regression, reducing the error between the reservoir output and the desired output signal. In most cases, this layer is simply a linear weighted combination of system states; however, more complex and non-linear readouts (and training methods) are possible.

The variety of intrinsic properties exploitable from physical systems—how they store and process information—is what makes these systems useful, powerful, and diverse. With physical systems, there are greater possibilities for faster, less-expensive, and more energy-efficient computing. This is in part due to less top-down design, fewer data conversion steps and constraints, removal of traditional data transfer bottlenecks, and architectures that tend to be more robust and less error-prone. However, there are still many gaps left in our understanding of what makes these systems compute.

In the remainder of this chapter, we highlight areas of unconventional computing to discuss and map out challenges and areas still requiring further development. We outline the importance of representation and instantiation in Sect. 3 and discuss the programming of reservoirs in Sect. 4. In the final section, we discuss a newly proposed framework to evaluate what makes a good computing substrate, providing a new perspective on how to build and compare physical reservoir computers.

2 Computing with Physical Systems

Biological systems vastly outperform certain aspects of classical computing paradigms, from possessing inherent fault-tolerance to forming highly parallel machinery. Much of this performance is achieved by exploiting physicality and embodiment (Stepney 2007), sharing and distributing computational effort throughout the system, from small-scale micro-organisms to large-scale swarms. Biological systems exploit physical interactions through feedback with the real world, utilising features such as morphology and direct and indirect changes to the environment. Many of these systems comprise simple elements (e.g., molecules and cells) that emerge and coalesce into more complex (e.g., multi-cell organisms and ecosystems), but robust, structural layers working across different spatial and temporal scales. Such grounding properties (and many more) have enabled these complex systems to thrive and evolve, adapting and co-evolving with their local ecosystem.

In modern science and engineering, there are many attempts to mimic the computational properties, efficiency, and behaviours of biological systems on conventional machines. In many ways, such attempts are flawed, or at least inefficient. Techniques and models attempt to imitate the performance of an embodied system in a non-embodied abstraction, in a process that requires an often cumbersome and inefficient transformation to a symbolic representation. In essence, such transformations detract from many of the physical aspects that make natural systems so powerful.

The limitations of conventional computing paradigms are well defined, and we are rapidly approaching the limitations of current CMOS technology (Lloyd 2000). For technology to move forward, many of these limitations need to be overcome, by using the same classical paradigms or alternative ones.

The conventional von Neumann computing architecture, based on the stored-program computer concept, although expertly refined over decades, has 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 design complexity.

2.1 Unconventional Computing

The field of *unconventional computing* has for many years attempted to address the limitations of conventional computing by providing alternative architectures, systems, and models that typically exploit the underlying physics and many-scale interactions of the real world. Many forms of unconventional systems now exist, from the mathematical reversible and chaos computing concepts to physical–chemical and neuromorphic computing. Recent collections of theory and practice of unconventional systems include (Adamatzky 2016a, b; Stepney et al. 2018).

Unconventional systems have a long history. Some systems link back to the mid-twentieth century cybernetics movement, such as Pask's experiments with electrochemical assemblages in ferrous sulphate solutions (Cariani 1993; Pask 1959). Others include Turing's unorganised machines (A- and B-type random networks) (Turing 1969) constructed from simple components in a random structure capable of learning. Even further back there are analog computing models and systems such as the differential analyser (early twentieth century), Babbage's Difference Engine (nineteenth century), slide rules, Orrerys (mechanical models of the solar system), and astronomical clocks.

Pask's work—a novel example of unconventional systems and methods at the time—was conducted exclusively in the physical domain and sought to evolve functional dendrite-like structures by passing current directly through immersed elec-

trodes. To alter the growth of these structures, he selected which electrodes to pass current between, with the resulting linkage conductance representing synaptic weights. In his experiments, Pask manually selected and "evolved" 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.

Turing's work, on the other hand, was more theoretically based, drawing on analogies with the human brain. What makes Turing's machines particularly interesting here is the clear analogies to the current reservoir computing paradigm.

In general, exploiting computation directly at the substrate level, as biological systems do, is expected to offer advantages over classical computing architectures, such as exploiting physical and material constraints that could offer solutions "for free", or at least computationally cheaper (Stepney 2008). Extracting computation from these systems and physically *programming* them, however, is challenging. Sometimes, this is further complicated by a desire for minimal abstraction: exploiting emergent physical phenomena directly whilst maintaining a level of programmability that enables the system to perform a variety of tasks.

Hybrid digital–analog computers potentially solve this programming problem, where the digital system is trained to extract computation performed implicitly from an analog substrate. This is typically where many physical reservoir computers align, with the readout and training often carried out in the digital domain. An excellent example of a programmable hybrid system is Mills' Kirchhoff–Lukasiewicz Machines (KLM) (Mills 1995) based on Rubel's computational model of analog computation (the extended analog computer Rubel 1993). In Mills' work, the KLM device is controlled by a specially designed vector of bits, referred to as the "overlay" (Mills 1995). This overlay, representing the semantic "program", is used to define 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* (Mills 2008), 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. Thus, it is an analogy of the program, rather than an algorithmic function to be implemented.

For analog computers, the program and architecture may be indistinguishable or inseparable: to program the machine requires a change in the machine architecture. However, placing additional layers within the machine's architecture may reduce the amount of change required; for example, adding a standardised reconfigurable "middle" layer or a trainable readout in reservoir computing.

2.2 Configuring Physical Systems to Compute

Conventional computers are designed to be substrate-independent, where a symbolic virtual "machine" is engineered into physical hardware. Yet, not every com-

puting problem requires abstraction to a symbolic virtual machine, e.g., filtering, control problems, and solving differential equations. The digital computing pipeline from physical-to-abstract-to-physical tends to consume much more power than an equivalent analog counterpart, be constrained by serial-processing, and present many vulnerabilities in terms of security and coding errors.

Material/substrate-based computers are machines in which some computational process, or physical mechanism, may be extracted from a behaviourally diverse 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 (Stepney 2008). 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.

Given some material has potential properties useful for computing, the question is whether they are extractable, and whether the system can be trained, configured, or engineered to consistently exploit these properties.

Before the first physical reservoir computers emerged, a computing paradigm reminiscent of Pask's self-assembling ear was developed for novel substrates. This conceptual idea was named "evolution *in materio*" (EiM) by Miller and Downing (2002). Inspired by Thompson's seminal work on "intrinsic" hardware evolution (Thompson 1997), the principal idea is to use artificial evolution as a search method to find configurations that directly exploit the computational properties of complex materials.

To date, work with EiM has explored liquid crystal substrates, composites of randomly dispersed carbon nanotubes mixed with polymers, carbon nanotubes mixed with liquid crystal, and networks of gold nanoparticles (Broersma et al. 2017; Harding and Miller 2004; Massey et al. 2016; Miller and Downing 2002). Each of these substrates is evolved to achieve interesting computational properties on a variety of specific tasks, without it being known exactly how best to program them to perform those tasks. Applying evolved mappings and external "control" signals, the unknown internal properties of the composites are configured to produce physical solutions to computational tasks.

Current progress of EiM and physical reservoir computing still remains at the substrate level: single devices/systems and simple architectures. This limits the complexity and types of tasks that are solved. Solving problems with increased complexity will require layers of (non-symbolic) abstraction, hierarchy, and possibly multi-substrate designs, whilst maintaining substrate-level exploitation and efficiency. How to implement and program such architectures is still unclear.

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

At the basic level, a generic methodology to characterise the substrate, suitable architectures, and higher-level constructs that naturally fit the substrate is missing.

For example, in EiM, often too little is known about the substrate being evolved; in principle this is the point, but in practice it limits its full potential. A minimal criterion for evolution to excel, or for reservoir computing to work, needs to be established. At the same time, the realistic potential of the substrate needs to be determined and categorised. This includes whether a substrate is compatible with other substrates, what role it can fulfil in a hierarchy or high-level program, and what other basic physical limitations exist.

3 Reservoir Computing with Physical Systems

A key advantage to using the reservoir computing paradigm is that there is no requirement to control individual elements within the system. This makes it applicable to many complex structures where the exact arrangement and manipulation of internal elements are 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 many cases, a physical substrate is computationally useful only when configured or perturbed. Therefore, forming a useful reservoir requires the tuning of physical parameters. This itself implies that a single substrate instance can realise a range of reservoirs of varying quality through different physical configurations.

Using the reservoir model, we argue that any substrate and subsequent configuration is represented by its abstract "quality" in a space of all possible reservoirs, and that this space is very different from the configuration space. Methods for "pre-training" reservoirs (or selecting appropriate parameter ranges) to navigate this reservoir space are therefore essential tools to find and discover functional, possibly optimal, reservoirs within all possible material states. Figure 1 shows a conceptual representation of a single substrate's space of all possible physical configurations and its abstract reservoir equivalence.

Different substrates have their own such spaces; some perhaps have no functioning reservoirs; some perhaps possess many configurations in distant regions of the space that produce ideal properties for reservoir computing. When substrate "richness" and complexity increases, these complex structured spaces have a greater probability to be "deceptive" and difficult to navigate. Configurations close to an ideal solution may themselves be computationally uninteresting; working configurations may be unstable or critical when perturbed by noise and other external signals.

To create and navigate these spaces, basic mechanisms must be defined: how to *encode* and *represent* information flowing both in and out of the system. Depending on this, each space will vary drastically. For example, the size and density of functional reservoirs in these spaces will increase or decrease given different encodings. The combination of multiple encodings and representations will also have an effect, possibly leading to even larger and richer spaces. A prime example of how encoding affects computability is the delay-based reservoir computing technique using a

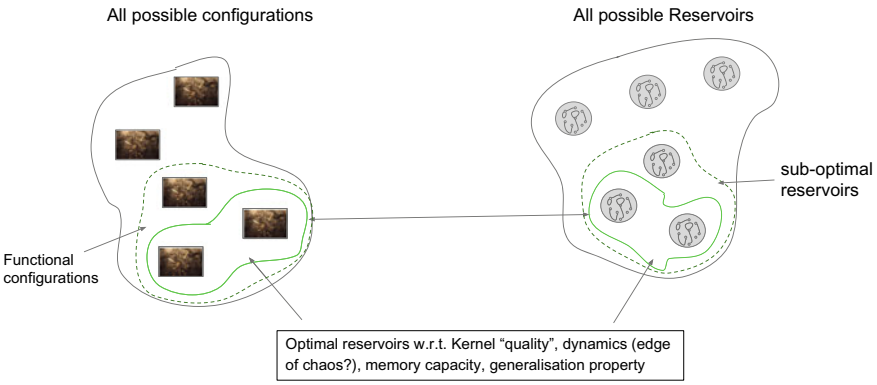


Fig. 1 Mapping between substrate configuration space and reservoir quality; from Dale (2015)

single non-linear node with delayed feedback, synonymous with optoelectronic and photonic systems (Appeltant et al. 2011, 2014; Brunner et al. 2013; Paquot et al. 2012). In these systems, the input encoding defines a network of virtual processing nodes in the time-domain rather than implementing a physical spatial network of nodes.

3.1 Encoding and Representation in Reservoir Computers

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 is challenging, if not impractical, to capture using a closed mathematical model. As such, trying to program these embodied systems requires different techniques.

Unconventional computers have the potential to be faster, and/or consume less power, but in order to do so requires that the model of computation naturally fits the material rather than imposing an inappropriate model that fights its implementation. As Caravelli and Carbajal (2018) describe it: "the problem needs to be specified using the language of the computing device. Using the wrong language increases the difficulty of the problem, and consequently decreases performance."

Most unconventional systems have only a primitive, or even no explicit, computational model. This makes it difficult to build higher-order representations, leaving the programming side heavily underdeveloped. As a result, programs are typically stuck at the equivalent assembly language level, with all the problems associated with a lack of abstraction and usability.

Representation and encoding are therefore critical to future progress; identifying what representations work best, and designing high-level programs that naturally fit the reservoir model, need to be further developed.

3.2 Abstraction/Representation (A/R) Theory

To discuss the encoding and representation problem for physical reservoirs, we first need to define as to when a physical system computes.

In order to distinguish when computation is occurring in a physical system, as opposed to the system simply just "doing its thing", Abstraction/Representation (A/R) theory has been developed (Horsman et al. 2014, 2017, 2018). In A/R theory, *physical computing is the use of a physical system to predict the outcome of an abstract transformation.*

The theory identifies objects in the domain of *physical systems* (including computers), the domain of *abstract objects* (including computational models), and the *representation relation* which links the two.

The *representation relation* is primitive and maps from physical to abstract objects, $\mathcal{R} : \mathbf{P} \rightarrow M$, where $\mathbf{P}$ is the set of physical objects, and M is the set of abstract objects. When two objects are connected by $\mathcal{R}$, we write them as $\mathcal{R} : \mathbf{p} \rightarrow m_{\mathbf{p}}$. The abstract object $m_{\mathbf{p}}$ is then said to be the *abstract representation* of the physical object $\mathbf{p}$. Instantiation is a map from abstract object to physical object: $\widetilde{\mathcal{R}} : M \rightarrow \mathbf{P}$. When two objects are connected by $\widetilde{\mathcal{R}}$, we write them as $\widetilde{\mathcal{R}} : m_{\mathbf{p}} \rightarrow \mathbf{p}$. The physical object $\mathbf{p}$ is then said to be the *physical instantiation* of the abstract object $m_{\mathbf{p}}$.

Abstract evolution maps abstract objects to abstract objects, which we write as $C : m_{\mathbf{p}} \rightarrow m'_{\mathbf{p}}$. If we instantiate $m_{\mathbf{p}}$ in $\mathbf{p}$, then its corresponding physical evolution map is given by $\mathbf{H} : \mathbf{p} \rightarrow \mathbf{p}'$. If we now apply $\mathcal{R}$ to the outcome state of the physical evolution, we get its abstract representation $m_{\mathbf{p}'}$. See Fig. 2.

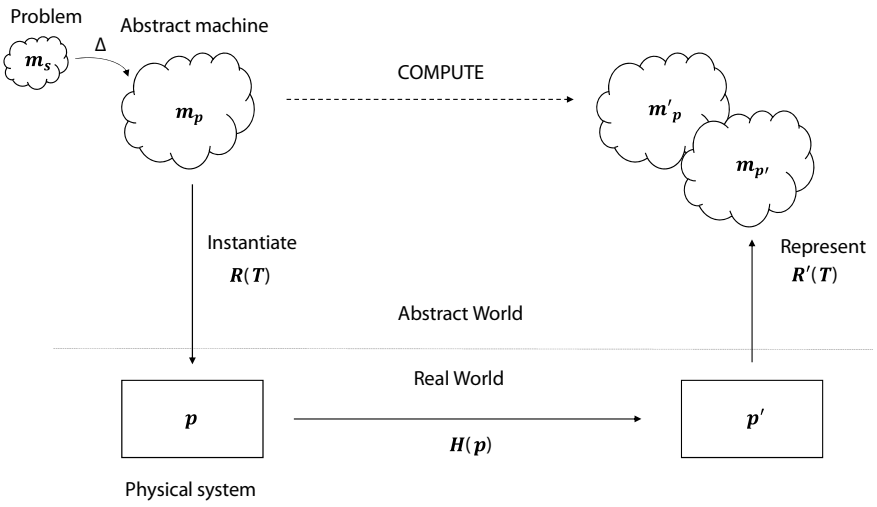


Fig. 2 Commuting diagram: mapping between abstract space and physical space. T represents appropriate theory for encoding and representation. Δ is translation of abstract problem into abstract machine. Other definitions are given in the text

We now have two abstract objects, $m'_{\mathbf{p}}$ and $m_{\mathbf{p}'}$. If these are *sufficiently close*, then we say that the physical system has computed the desired result of the abstract computation. For a *well-engineered* system, one where we can rely on this sufficient closeness, we no longer have to compare the two answers, and can take $m_{\mathbf{p}'}$ as the prediction of the desired result of the computation $m'_{\mathbf{p}}$.

To extend the theory to multiple physical systems, as many practical unconventional devices will most likely consist of, the *heterotic* computing framework has been developed (Horsman 2015; Kendon et al. 2011, 2015). The term "heterotic" captures the "hybrid vigour" of complex systems, where the whole is greater than the sum of its parts. Possible heterotic systems could consist of multiple devices and different computational models combined to create more expressive and programmable computers.

The concept of hybrid reservoir systems, "mixing and matching" different unconventional systems, both virtual and physical, is largely unexplored in the literature. However, on the surface, it looks promising for further research as the benefits of hierarchical reservoir systems come into light (Dale 2018a; Gallicchio et al. 2017).

3.3 Observing Reservoir States

Here, we define our representation $\mathcal{R}$ of physical system states. To interpret a physical substrate as a reservoir, the definition in Konkoli et al. (2018) is adapted, where the observed reservoir states $x(n)$ form a combination of the substrate's implicit function and its *discrete* observation:

$$x(n) = \Omega(\mathcal{E}(W_{\text{in}}u(t), u_{\text{config}}(t))) \tag{1}$$

where $\Omega(n)$ is the observation of the substrate's macroscopic behaviour and $\mathcal{E}(t)$ the continuous microscopic substrate function when driven by the input $u(t)$. W_{in} symbolises a set of input weights common in all reservoir systems. The variable $u_{\text{config}}(t)$ represents the substrate's configuration, whether that be through external control, an input–output mapping, or other method of configuration. Equation 1 is a simple case where no feedback is returned to the system. To add feedback, the input variables y and feedback weights W_{fb} are added to $\mathcal{E}(.)$.

This formalisation of the reservoir states separates the system into contributing parts, including the observation and configuration method, which as a whole represents the overall physical reservoir computer.

An exploitable feature of this definition is that there is no need for the input mechanism to be the same as the observation method. For example, an input may be encoded as an electrical signal, and the reservoir states observed as deformations in physical structure. This additional channel of communication leads to an increase in system bandwidth to exploit, allowing multiple measurement and stimulation techniques to be used in tandem.

In Eq. 1, note that the intrinsic substrate function ($\mathcal{E}$) is presumed to be fixed, clamped, or set by u_{config}. However, depending on the system, $\mathcal{E}$ may change when interacted with or observed (commonly known as the "observer" effect), and may therefore be non-deterministic.

When physical systems are connected to stimulation and recording equipment, it is critical not to overlook that the whole interface will, in some way, affect the computing process, which may or may not be included in the model. In most experimental works, it is imperative that the substrate being exploited is sufficiently isolated from the rest of the system, e.g., from its controlling equipment and its environment. In Dale (2018b), it was shown how evolution could find ways to include and exploit the interfacing equipment as part of the reservoir system. Despite its positive contribution, generally improving performance, the dynamics of the substrate became less important to the computing system. For cases such as this, we include the Ω function in Eq. 1, as the observation process itself may affect the system output.

This phenomenon, unique to physical substrates, is a trade-off problem, presenting itself more within certain systems. If the substrate is fixed permanently to the same measuring equipment, the contribution of the interface to the overall "computation" is 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.

The final output $y(n)$, the last part of the system, is determined by the readout function g on the observation $x(n)$:

$$y(n) = g(x(n)) \tag{2}$$

In many reservoir systems, the readout function g tends to be a linear weighted combination of system states for RC, and once trained remains static.

As previously mentioned, the readout layer can be implemented within a different medium or domain from the reservoir substrate itself. For example, an analog material may be combined with a digital readout layer. This flexibility adds extra programmability, and even compatibility with other devices, but may come with specific costs and benefits.

At present, methods that take advantage of $u_{\text{config}}(t)$ and $g(x(n), n)$ changing with time are uncommon. Although this comes at the price of complexity, abstract programs where both implement functional primitives, providing higher-order representations in $y(n)$, are theoretically possible.

4 The Search for Reservoirs

The classical approach to programming and manipulating physical systems requires (to some extent) a good understanding of the properties and interactions within that system. This results in the traditional top-down programming approach. Reservoir computing and other "intelligent" systems apply alternative mechanisms, e.g.,

through training, learning, and heuristics. For each of these programming approaches, the details of the system are exploited in a controlled manner typically without explicit instructions from a human. For example, the approach has learned by itself where best to clamp global or local dynamics using a bias signal, or the exact strength and combination of control signals required to exploit and alter the structure.

The actual "programs" implemented by the substrate and the system as a whole, however, are different. The previous approaches therefore act as meta-programming approaches. Following the previous discussion, we can think of these programs as containing details about the encoding and representation of the system and its physical parameters. Different programs may exploit the same physical phenomena (e.g., electromagnetism, electrochemical, and optoelectronic) but encode and represent data using only specific physical parameters (e.g., electric fields), or, alternatively, use different encodings and physical parameters (e.g., magnetic fields and chemical reactions). This can lead to a combinatorically vast search space of potential computer programs, e.g., reservoirs on a single device.

Reservoir substrates typically possess fixed dynamics after applying carefully selected parameters and parameter ranges to exhibit desirable dynamical properties. Echo State Networks are an exemplar of this tuning problem, requiring global scaling parameters to induce specific dynamics, such as possessing the echo state property (Jaeger 2001a). With many unconventional systems, the parameter space is complex and therefore difficult to navigate. As complexity increases, simple (unconstrained) random configurations often result in sub-optimal performance, as the probability of stumbling upon optimal parameters is low. In general, no matter the substrate, good parameter selection and encoding—the basic program—are essential to produce high-performing reservoir systems.

In the RC community, many meta-programming approaches have been used to optimise the parameters of virtual reservoirs. For example, techniques include particle swarm optimisation (Basterrech et al. 2014; Sergio and Ludermir 2012), Bayesian optimisation (Yperman and Becker 2016), gradient-based information (Yuenyong 2016), multi-objective optimisation (Krause et al. 2010), and evolutionary algorithms (Chatzidimitriou and Mitkas 2010; Ferreira and Ludermir 2011; Jiang et al. 2008; Matzner 2017; Qiao et al. 2017). Other heuristics continue to be added; see the recent survey (Abubakar et al. 2018).

These approaches are simple to apply due to the RC framework's training partition. What often separates reservoir computers from other learning systems is this division in training. Reservoirs are typically generated or configured (partially programmed) using one technique, and the readout is trained using another (fully programmed), avoiding challenges when training complex networks.

Using this separation, a physical substrate can be partially programmed, then quickly retrained/programmed to solve different tasks by manipulating only the readout. This extra level of programmability provides advantages in time to train and sometimes performance.

The typical focus when programming is to optimise the substrate to a specific task. This approach provides little insight into the wider computing ability of such substrates, however, such as its bottlenecks, strengths, and weaknesses, its "sweet-

spots", etc. An optimal configuration may solve a specific problem very well, but by itself, be uninteresting. Exploring the configuration space and abstract space of all reservoirs/programs could uncover more general characteristics of the system, and inform us more about what makes it compute and what is computable.

To map large complex spaces, typically with vast numbers of parameters (e.g., for physical substrates or deep/hierarchical structures) requires efficient search methods, rather than random or grid search. This is where a new class of *open*-ended *Quality Diversity* (QD) algorithms (Pugh et al. 2016) could have a significant impact, improving design, understanding, and exploration of new physical computing devices. These algorithms have experienced great success in the embedded setting of evolutionary robotics (Mouret and Doncieux 2009, 2012) and hard exploration problems (Ecoffet et al. 2019). These algorithms include novelty search with local competition (NSLC) (Lehman and Stanley 2011), and multi-dimensional archive of phenotypic elites (MAP-elites) (Mouret and Clune 2015).

The basic concept is to define a low-dimensional feature space, or *behaviour* space, separate from the high-dimensional parameter space, within which performance and behaviour are explored rather than optimised directly. (However, these algorithms can also be used to optimise directly, making them a superset of optimisation algorithms Mouret and Clune 2015.) During the search process, novel behaviours represent markers—points to seed or diverge from, or elites that represent the best behaviours in the low-dimensional search space, promoting the discovery of new solutions that would be otherwise too difficult to reach using standard optimisation algorithms.

These particular algorithms specialise in domains where the fitness landscape—the space of all solutions with respect to performance— is often deceptive and sparse. This deceptiveness, in evolutionary algorithm terms, typically relates to the *indirect* encoding between the genotype (encoding) and phenotype (physical instantiation), where information in the genome may affect many parts of the phenotype.

They overcome deceptive spaces in part by tuning the trade-off between exploration and exploitation, promoting diverse solutions as well as opimising local niches. As a by-product, the process produces a holistic view of the entire search space, rather than a single optimised solution. Mouret and Clune refer to these algorithms as a new class of "Illumination algorithms" (Mouret and Clune 2015), mapping the highest performing solutions in each region of the feature space and determining each region's overall potential.

In summary, the programming of single substrates for reservoir computing is sometimes simple, or complex. It is simple when there is a relatively good understanding of the substrate parameters, but considerably more complicated when little is known about parameters and resulting computational properties.

It is important to remember that a physical system's program encompasses encoding as well as representation, which may be altered. This in turn results in an enormous range of potential programs even for a single substrate.

As architectures move from single devices to multiple devices, programming will become increasingly challenging. To fully utilise such architectures, we first need to understand more about the computing capabilities of substrates. To do this, we have

mentioned one class of search algorithms that focuses on characterising the space of all reservoirs implemented by a single substrate rather than optimising parameters towards specific tasks.

5 What Makes a Good Physical Reservoir?

The ability to perform useful information processing is an almost universal characteristic of dynamical systems, provided a fading memory and linearly independent internal variables are present (Dambre et al. 2012). However, each dynamical system will tend to suit different tasks, with only some performing well across a range of tasks.

In terms of all reservoirs realisable by a substrate, many may perform well, given the selection of suitable parameters. Yet, many will also be unstable and unusable. This selection process may be easy for well-understood and constrained systems, but poses a serious challenge when less is known about the substrate and how it will react to stimulation, or different parameter ranges and encodings.

Until recently, no experimental framework has tried to explore, map, and compare the complete computational expressiveness—including encodings and representations—of physical and virtual substrates. That is, no practical method has been proposed to characterise the substrate and its individual instantiations to build a comprehensive view of the computing *quality* of substrates. A/R theory, Sect. 3.2, is a framework to *define* physical computing, but it does not provide guidance on how to determine an appropriate encoding, instantiation, or representation.

To tackle this non-trivial problem, we have developed the CHAracterisation of Reservoir Computers (CHARC) framework (Dale et al. 2019b). The framework describes characterisation phases and adaptable levels to measure *quality*, defined as the total capacity to realise distinct reservoirs in terms of different dynamical properties.

5.1 Framework Outline

To characterise a test substrate, two phases must be completed: quality assessment of a *reference* substrate (phase one), and characterisation of the *test* substrate (phase two). Phase one provides something to compare to and is typically carried out only once, provided a suitable reference is chosen. The basic structure and flow of each phase are shown in Fig. 3.

The first level is the *definition* level (Fig. 3, step 1). Here, the *behaviour space* of all abstract reservoirs is defined. This abstract space represents the dynamical behaviour of the substrate when configured. To create this n-dimensional space, n independent property measures are identified and used. Each point in this space is a behavioural representation of a substrate's configuration according to different measures.

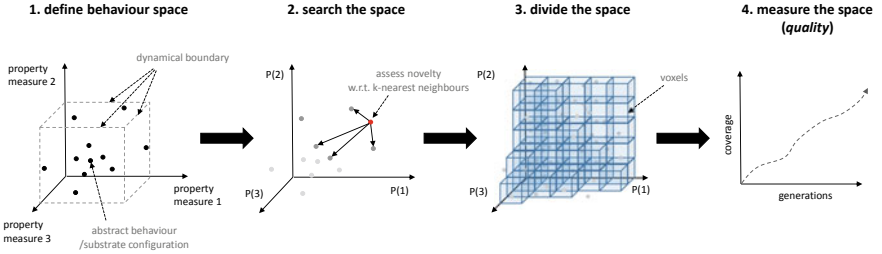


Fig. 3 CHARC framework workflow. This is typically carried out for both phases, using the same framework meta-parameters

More measures of distinct properties should result in greater model reliability and therefore greater confidence in the quality measure. However, when defining the behaviour space some properties are difficult, if not impossible, to measure across all substrates. Any measure used with the framework should represent the observed behaviour of the system, rather than specific properties related to its implementation, or measures unpractical/impossible to carry out with physical and/or black-box systems.

To demonstrate the basic framework structure, three measures are applied in Dale et al. (2019b) to define the behaviour space: Kernel Rank, Generalisation Rank, and Memory Capacity.

Kernel rank (KR) measures the reservoir's ability to produce a rich non-linear representation of the input. Generalisation rank (GR) is a measure of the reservoir's capability to generalise given similar input streams. More information about these measures can be found in Legenstein and Maass (2007). Low ranking values in both measures typically represent a system in an ordered regime, and both having high values equate to chaotic regimes. According to Büsing et al. (2010), the most interesting reservoirs are found to exhibit a high kernel rank and a low generalisation rank. However, in terms of matching reservoir dynamics to specific tasks, the right balance varies.

The measure for memory capacity (MC) in Dale et al. (2019b) captures the linear short-term memory capacity of a reservoir. The linear measure was first outlined in Jaeger (2001b) 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. As demonstrated in Dambre et al. (2012), other measures quantifying the non-linear, quadratic, and cross-memory capacities are possible, providing a more complete picture of memory in dynamical systems.

As explained in Dale et al. (2019b), the three selected measures are important, but by themselves do not capture every interesting dynamical property of dynamical systems. Therefore, more measures are still desired. For example, the total capacity

measures defined in Dambre et al. (2012) could replace, or add to, the axes of the behaviour space.

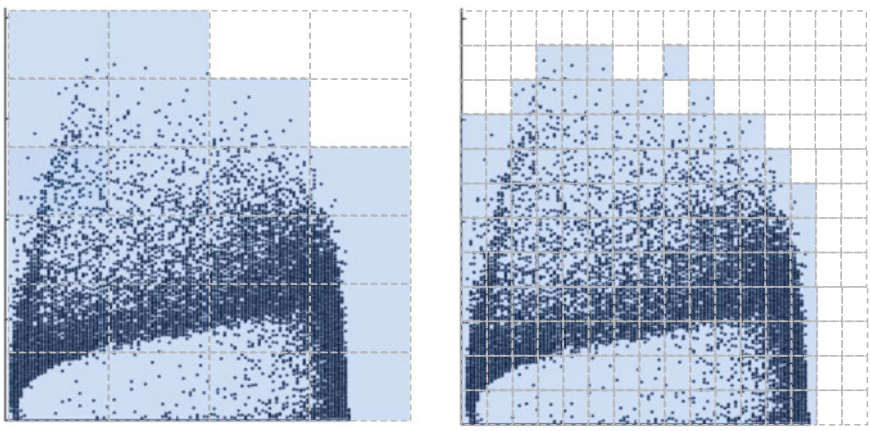
The *Exploration* level (Fig. 3, step 2) encompasses the search method and mapping process. At this level, the mapping between abstract reservoir and substrate configuration is explored.

Exploration is carried out in the behaviour space using an implementation of novelty search (Lehman and Stanley 2008). Novelty search, an open-ended genetic algorithm, navigates the behaviour space searching for novel solutions until some user-defined termination point, e.g., after so many evolutionary evaluations, or possibly when the rate of exploration has stalled.

Given enough time (search evaluations), the exploration process can outline the boundaries of the system dynamics, i.e., the boundary defining what behaviours are possible and not possible. In Dale et al. (2019b), this is demonstrated for a physical carbon nanotube composite. Due to the limited behavioural freedom of that substrate, its boundaries are relatively constrained, showing that any tasks requiring more than minimal memory requirements tend to be unsuited to the substrate. The results also demonstrate that exploration can be used to identify the practical use, if any, of the substrate, or whether the selected method of encoding, representation, and configuration is appropriate.

The final *Evaluation* level (Fig. 3, steps 3, 4) defines the mechanisms to evaluate quality. To measure quality, the behaviour space is divided into voxels/cells. Figure 4 demonstrates how the behaviour space may be divided and measured depending on the experimentally defined quality *resolution*.

The number and size of voxels/cells depend on the spaces being compared. For example, the voxel size can be large and coarse, grouping many local behaviours,



(a) Low resolution (b) High resolution

Fig. 4 Example of a 2-D behaviour space divided into different voxel sizes. Each dot represents a measured behaviour in a generic 2-D behaviour space. Squares represent 2-D voxels

signifying the extent of the region of behaviours discoverable (Fig. 4a), or an increase in resolution to provide a more fine-grained view of how the space is filled (Fig. 4b).

Overall, the total number of voxels/cells occupied forms the measure of quality. The quality value therefore represents an approximation of the system's dynamical freedom, or the substrate's capacity to instantiate different distinct reservoirs.

5.2 *Characterising Substrate Quality*

The initial evaluation and validation of the CHARC framework (Dale et al. 2019b) assess the quality of three very different dynamical systems: a recurrent neural network, a numerical simulation of a Mackey–Glass oscillator with a delay line, and a physical carbon nanotube-based substrate.

Artificial recurrent neural networks such as echo state networks (ESN) are considered state-of-the-art reservoir substrates, so ESNs of various sizes form the reference substrate. The simulated Mackey–Glass oscillator with a delay line (referred to as DR in this chapter) is a reservoir substrate known to perform remarkably well across different reservoir computing benchmarks (Appeltant et al. 2011; Paquot et al. 2012; Duport et al. 2012). The physical substrate comprising a mixed carbon nanotube–polymer (poly-butyl-methacrylate) composite (CNT) is known to perform well on several simple computational tasks (Mohid et al. 2016; Dale et al. 2016b, 2017), but struggles with some harder reservoir computing benchmarks (Dale et al. 2016a).

Results in Dale et al. (2019b) show the behavioural freedom of each substrate is significantly different when compared using a voxel size of $10 \times 10 \times 10$ behavioural units. The quality of each substrate is measured as the total number of voxels occupied after each experimental run of 2000 search generations. Figure 5 shows the number of voxels occupied as the exploration process progresses (every 200 generations), with error bars displaying the min-max values for 10 independent evolutionary runs.

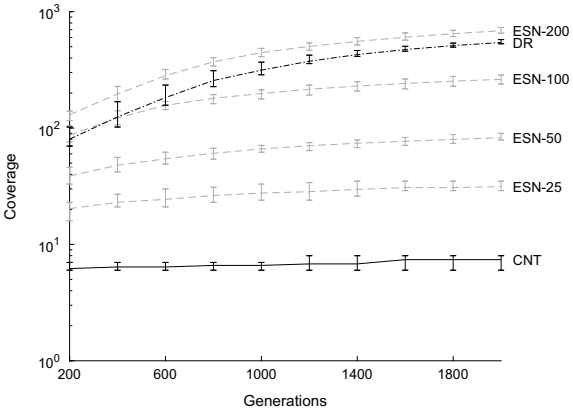
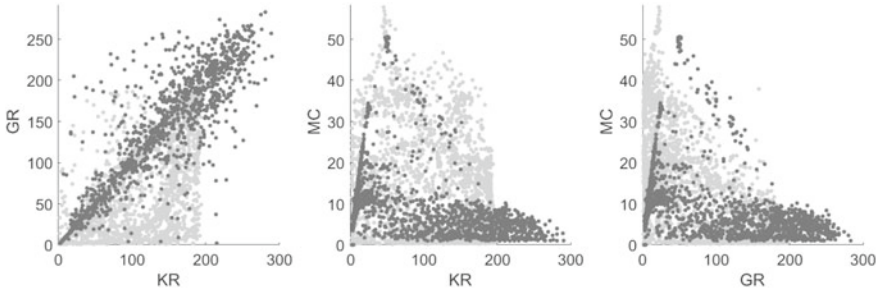
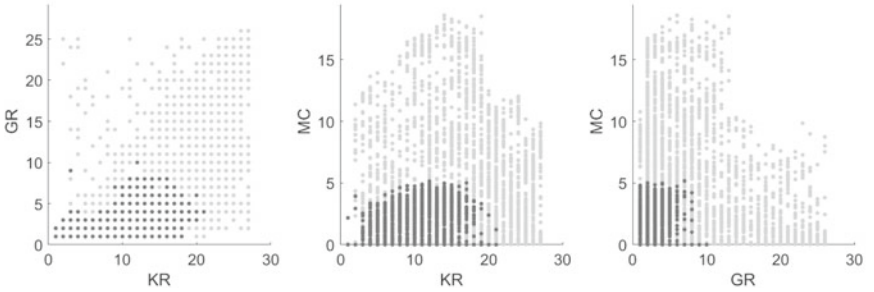


Fig. 5 Voxel measure of coverage as the number of search generations increases. Test substrates are shown as solid black lines (DR, CNT), and reference substrates (ESNs) are dashed grey lines. Error bars display the min-max values for all search runs. Note the logarithmic coverage scale



(a) 200 node ESN (light grey) with DR (dark grey)



(b) 25 node ESN (light grey) with CNT (dark grey)

Fig. 6 Behaviours discovered when exploring the ESN, CNT, and DR substrates. To visually compare substrates, each test substrate is plotted over the reference substrate with the most similar quality

The largest measured ESN networks (200 nodes) have the highest quality. The DR substrate (with 400 virtual nodes) is closely equivalent to the 200 node ESN substrate in terms of quality. The CNT substrate, on the other hand, had a considerably smaller quality than an ESN of just 25 nodes.

The quality value alone gives only a single numerical representation of quality. It is valuable to perform a visual inspection of the explored behaviour space to get a more nuanced view of a substrate's performance.

The discovered behaviours of the DR and CNT substrates are shown in Fig. 6. In the plots, the behaviours for each test substrate are presented in the foreground and the reference substrate (ESN) with the most similar quality (200 node ESN in Fig. 6a and 25 node ESN in Fig. 6b) in the background.

The DR behaviours (Fig. 6a) extend into regions that the 200 node ESN cannot reach, resulting in what appears to be fewer occupied regions than the ESN. However, this does not imply that such regions cannot be occupied. Given more search generations, these regions would likely be filled, as similar behaviours are already discovered.

The CNT substrate (Fig. fig:bsofsubstratesb) struggles to exhibit enough (stable) internal activity to create a strong non-linear projection, and to effectively store recent

input and state information, agreeing with previous results (Dale et al. 2016a, b, 2017). This suggests why only a limited range of tasks are suitable for the substrate, and why small ESNs tend to be good models of the substrate.

Overall, the results of the CNT substrate show that, using its current encoding and representation, only a limited set of behaviours is possible. In future work, quality might be improved using other types of input stimulus and control signals, or reading/combining other electrical output signals.

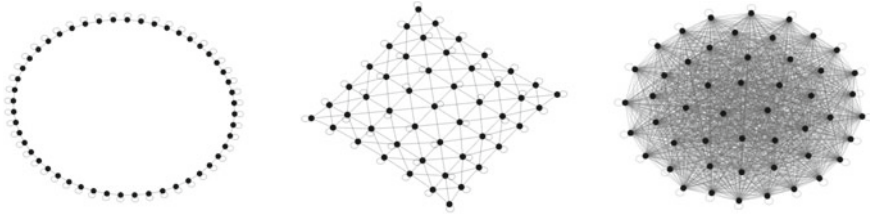
5.3 Using Quality to Assess Substrate Design

In this section, we demonstrate how the CHARC framework can be used to assess, and potentially inform, the design of future reservoir substrates.

In Rodan and Tiňo (2010) and Rodan and Tino (2011), it is shown that simple and deterministic connection topologies tend to perform as well as, or better than, standard (fully connected) randomly generated reservoir networks on a number of benchmark tasks. If the design of state-of-the-art reservoirs requires only simple structures and topologies, physical reservoirs become much easier to design and implement.

To investigate what effect structure and connection complexity have on dynamical behaviour, CHARC has been applied to neural graphs of varying complexities and sizes (Dale et al. 2019a). The effect of network topology and connection complexity (in this case, *directed* and *undirected* graphs) is investigated by evaluating three simulated recurrent network topologies: *ring*, *lattice*, and *fully connected* networks.

The ring topology (Fig. 7a), with the least complexity, has nodes with a single self-connection and one connection to each of its direct neighbours. The basic ring topology is the simplest network to implement in physical hardware as the number of connections required is small. Ring structures have already been applied to many reservoir computing systems, including minimum complexity echo state networks (ESN) (Rodan and Tino 2011), DNA reservoirs (deoxyribozyme oscillators) (Goudarzi et al. 2013), Cycle reservoirs with regular jumps (CRJ) (Rodan and



(a) ring (b) lattice (c) fully-connected

Fig. 7 Network structures investigated in Dale et al. (2019a)

Tiňo 2010), and delay-based reservoirs using a single non-linear node with a delay line (Appeltant et al. 2011; Paquot et al. 2012).

The lattice topology (Fig. 7b) has greater connection complexity. In Dale et al. (2019a), a square grid of neurons with each connected to its Moore neighbourhood is defined. In this configuration, each node (except for the perimeter nodes) has eight connections to neighbours and one self-connection, resulting in a maximum of nine connections per node. Lattice networks/models are common in physics. Examples include discrete lattices like the Ising model with variables representing magnetic dipole moments of atomic spins, and the Gray–Scott reaction–diffusion model to simulate chemical systems (Pearson 1993). Physical substrates often have a regular grid of connections. Lattice networks are therefore more realistic representations of many physical systems that would be considered for reservoir computing.

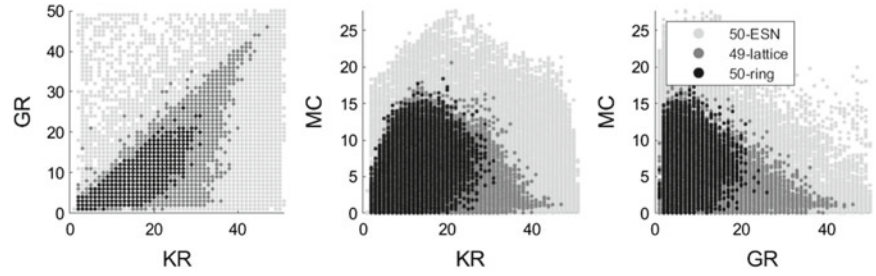
The fully connected topology (Fig. 7c) has the most connections and is considered the most complex. This type of network is challenging to implement in physical hardware.

The results in Dale et al. (2019a) show that a directed and fully connected topology (the ESN network) occupies a greater area of the behaviour space, possessing a higher quality than the others, independent of size. The other topologies struggle to reach similar levels of coverage. The behavioural limits of these simpler networks appear to have been reached.

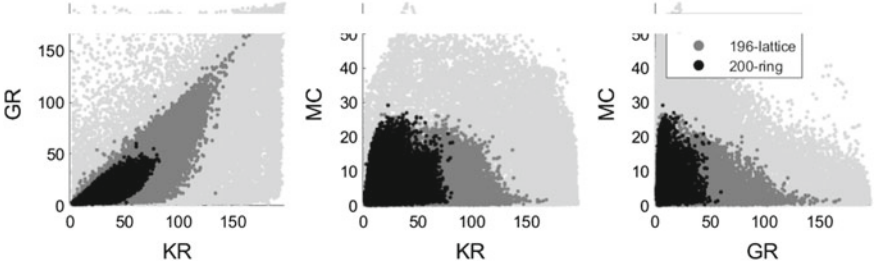
A common pattern found across all topologies and connection types is that quality increases with network size. Simpler structures can produce similar quality to more complex networks simply by increasing network size. A visualisation of how two network sizes (50 and 200-neuron) cover the behaviour space using each network topology is given in Fig. 8. ESNs tend to occupy regions the others cannot, such as chaotic regions (both high KR and high GR), and regions with larger memory capacities. The ring and the lattice topologies appear to have similar maximum memory capacities; however, lattices typically exhibit greater non-linearity and chaotic behaviour (higher KR and GR values) than rings.

Directed and undirected connection types are also assessed in Dale et al. (2019a). Connection type is equally as important as network topology. The coverage of a 100-neuron ring and lattice is shown in Fig. 9. Each plot shows the directed connection type (grey) and undirected type (black). Directed connections typically result in broader dynamical behaviour, producing more "challenging" behaviours (high KR and high MC). The difficulty in producing such behaviours exists because non-linearity (KR) and ordered dynamics (MC) are often conflicting.

Adding more accessible parameters (i.e., more connections) does not always lead to more dynamical behaviours. Networks with fewer free parameters (e.g., directed rings) could still produce similar qualities to more structurally complex networks (e.g., undirected lattices). How weights are structured and *directed* has a much greater effect on the quality of the network (Dale et al. 2019a), supporting similar results testing different hierarchical structures for reservoir computing (Dale 2018a).



(a) 50-neuron



(b) 200-neuron

Fig. 8 Topology effects. Superimposed behaviour space of **a** 50-neuron network, **b** 200 neuron network. Showing 10 runs of 2000 generations each. Only the directed topologies are shown. From Dale et al. (2019a)

5.4 CHARC Conclusions

The CHARC framework offers techniques to address several significant challenges outlined for reservoir computing (Goudarzi and Teuscher 2016). It provides (i) a framework to describe the computational expressiveness and complexity of substrates as a function of parameters and behaviours, (ii) a means to assess whether measures better represent the qualities/properties of the system, and (iii) a measure to assess the limits of physical RC substrates.

As shown in Dale et al. (2019a, b), the framework has diverse applications. For example, it can be used to evaluate and compare different substrates and design choices, such as the role of structure in RC systems, without needing extensive testing on tasks (see Sect. 5.3). In Dale et al. (2019b), the framework is also used to model the non-trivial relationships between reservoir properties and performance. The model can reliably predict task performance based on behaviour, across different substrates, without the need to evaluate tasks directly on the substrate.

The framework could be used in evaluating and determining the benefits of physical RC implementations against non-physical ones, a question raised in Goudarzi and Teuscher (2016). The framework could be used to improve substrate design, or extended to other models of computation (Stepney 2019; Dale et al. 2020). The

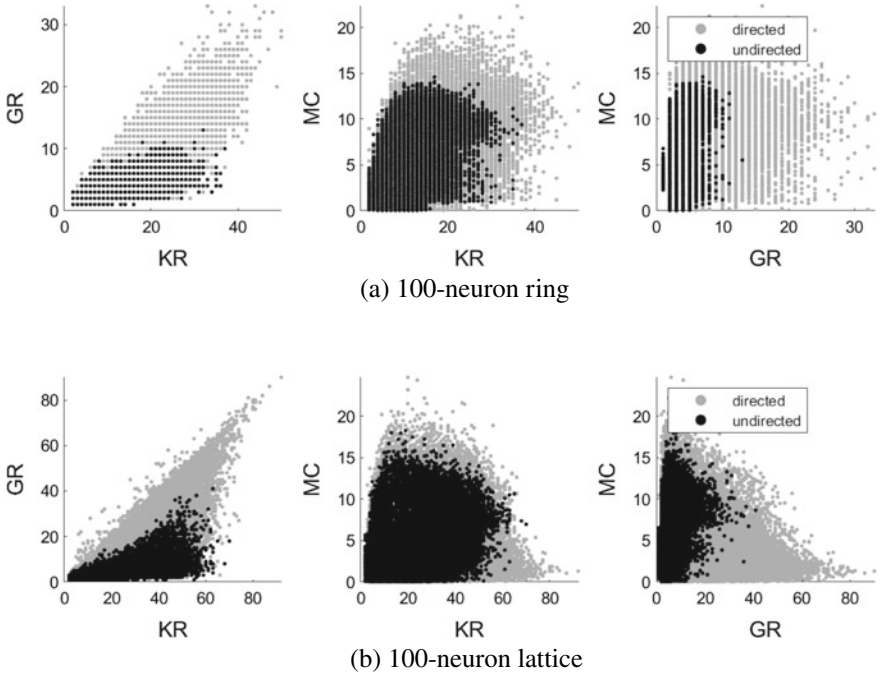


Fig. 9 Directed versus undirected topology effects. Superimposed behaviour space of 100-neuron ring and lattice network. Showing 10 runs of 2000 generations each. From Dale et al. (2019a)

flexibility of the framework allows for iterative improvements and expansion, for example, extending it to evaluate hybrid models using heterotic computing. With hybrid models, the behaviour space may be used as a repertoire of functional primitives, or modules, to build or program complex architectures, such as hierarchical reservoirs (Bürger et al. 2015; Dale 2018a; Gallicchio et al. 2017).

6 Conclusion

In a field that continues to diversify and excel, basic questions and theory in reservoir computing, and computing with physical systems, are being left behind. The rapid shift in recent years from virtual reservoir systems to physical ones has resulted in fundamental theory struggling to keep up with practice, with experiments highlighting the lack in theory, rather than theory proposing experiments.

In this chapter, we have highlighted areas of fundamental theory still undeveloped in computing with physical systems, and challenges in building unconventional physical [reservoir] systems. We have discussed the importance of appropriate encodings and representation, e.g., whether the computational model naturally fits

the computational substrate; programming and parameter selection, including thinking beyond single-task optimisation. And we have described a recent framework that provides some useful techniques to understand physical and reservoir systems, including a method to assess and compare the expressiveness of computing substrates, an exploratory method to configure systems when little is known about internal workings, a means to understand how substrate design choices affect behavioural/reservoir range, and how the abstract behaviour space maps to the physical space.

At present, reservoir computing is not a complete model for physical computing, however, it does have fundamental properties that make it simple, versatile, and highly efficient. Physical RC is therefore a *stepping stone* towards a more mature, principled general methodology for discovering and exploiting good natural computational models for material and physical computing.

Acknowledgements We thank the anonymous referees for their constructive comments, which helped improve this chapter. This work was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) SpInspired project [grant number EP/R032823/1].

References

B. Abubakar, I. Idris, I. Rosdiazli, M.S. Sadiq, Applications of metaheuristics in reservoir computing techniques: a review. IEEE Access **6**, 58012–58029 (2018)

A. Adamatzky (ed.), *Advances in Unconventional Computing: Volume 1: Theory* (Springer, 2016a)

A. Adamatzky (ed.), *Advances in Unconventional Computing: Volume 2 Prototypes, Models and Algorithms* (Springer, 2016b)

L. Appeltant, M.C. Soriano, G. Van der Sande, J. Danckaert, S. Massar, J. Dambre, B. Schrauwen, C.R. Mirasso, I. Fischer, Information processing using a single dynamical node as complex system. Nat. Commun. **2**, 468 (2011)

L. Appeltant, G. Van der Sande, J. Danckaert, I. Fischer, Constructing optimized binary masks for reservoir computing with delay systems. Sci. Rep. **4**, 3629 (2014)

S. Basterrech, E. Alba, V. Snášel, An experimental analysis of the echo state network initialization using the particle swarm optimization, in *Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC 2014)* (IEEE, 2014), pp. 214–219

H. Broersma, J.F. Miller, S. Nichele, Computational matter: evolving computational functions in nanoscale materials, in *Advances in Unconventional Computing* (Springer, 2017), pp. 397–428

D. Brunner, M.C. Soriano, C.R. Mirasso, I. Fischer, Parallel photonic information processing at gigabyte per second data rates using transient states. Nat. Commun. **4**, 1364 (2013)

J. Bürger, A. Goudarzi, D. Stefanovic, C. Teuscher, Hierarchical composition of memristive networks for real-time computing, in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)* (IEEE, 2015), pp. 33–38

L. Büsing, B. Schrauwen, R. Legenstein, Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. Neural Comput. **22**(5), 1272–1311 (2010)

F. Caravelli, J. Carbajal, Memristors for the curious outsiders. Technologies **6**(4), 118 (2018)

P. Cariani, To evolve an ear. Epistemological implications of Gordon Pask's electrochemical devices. Syst. Res. **10**(3), 19–33 (1993)

K.C. Chatzidimitriou, P.A. Mitkas, A NEAT way for evolving echo state networks, in *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)* (IOS Press, 2010), pp. 909–914

M. Dale, Unconventional reservoir computers: exploiting materials to perform computation, in *Eighth York Doctoral Symposium on Computer Science & Electronics* (2015), p. 69
M. Dale, Neuroevolution of hierarchical reservoir computers, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2018)* (ACM, 2018a), pp. 410–417
M. Dale, Reservoir computing in materio. PhD thesis, University of York (2018b)
M. Dale, J.F. Miller, S. Stepney, M.A. Trefzer, Evolving carbon nanotube reservoir computers, in *International Conference on Unconventional Computation and Natural Computation (UCNC 2016)* (Springer, 2016a), pp. 49–61
M. Dale, J.F. Miller, S. Stepney, M.A. Trefzer, Reservoir computing in materio: an evaluation of configuration through evolution, in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (2016b), pp. 1–8
M. Dale, J.F. Miller, S. Stepney, M.A. Trefzer, Reservoir computing in materio: a computational framework for in materio computing, in *International Joint Conference on Neural Networks (IJCNN 2017)* (2017), pp. 2178–2185
M. Dale, J. Dewhirst, S. O'Keefe, A. Sebald, S. Stepney, M.A. Trefzer, The role of structure and complexity on reservoir computing quality, in *International Conference on Unconventional Computation and Natural Computation (UCNC 2019)*. LNCS, vol. 11493 (Springer, 2019a)
M. Dale, J.F. Miller, S. Stepney, M.A. Trefzer, A substrate-independent framework to characterise reservoir computers. Proc. R. Soc. A **475**(2226), 20180723 (2019b)
M. Dale, S. Stepney, M. Trefzer, Designing computational substrates using open-ended evolution, in *Artificial Life Conference Proceedings* (MIT Press, 2020), pp. 665–667
J. Dambre, D. Verstraeten, B. Schrauwen, S. Massar, Information processing capacity of dynamical systems. Sci. Rep. **2**, 514 (2012)
F. Duport, B. Schneider, A. Smerieri, M. Haelterman, S. Massar, All-optical reservoir computing. Opt. Express **20**(20), 22783–22795 (2012)
A. Ecoffet, J. Huizinga, J. Lehman, K.O. Stanley, J. Clune, Go-explore: a new approach for hard-exploration problems (2019), arXiv:1901.10995
A.A. Ferreira, T.B. Ludermir, Comparing evolutionary methods for reservoir computing pre-training, in *International Joint Conference on Neural Networks (IJCNN 2011)* (IEEE, 2011), pp. 283–290
C. Gallicchio, A. Micheli, L. Pedrelli, Deep reservoir computing: a critical experimental analysis. Neurocomputing **268**, 87–99 (2017)
A. Goudarzi, C. Teuscher, Reservoir computing: Quo vadis? in *Proceedings of the 3rd ACM International Conference on Nanoscale Computing and Communication* (ACM, 2016), p. 13
A. Goudarzi, M.R. Lakin, D. Stefanovic, DNA reservoir computing: A novel molecular computing approach, in *DNA Computing and Molecular Programming (DNA 2013)* (Springer, 2013), pp. 76–89
S. Harding, J.F. Miller, Evolution in materio: Initial experiments with liquid crystal, in *2004 NASA/DoD Conference on Evolvable Hardware* (IEEE, 2004), pp. 298–305
C. Horsman, S. Stepney, R.C. Wagner, V. Kendon, When does a physical system compute? Proc. R. Soc. A **470**(2169), 20140182 (2014)
D.C. Horsman, Abstraction/representation theory for heterotic physical computing. Philos. Trans. R. Soc. A **373**(2046), 20140224 (2015)
D. Horsman, S. Stepney, V. Kendon, The natural science of computation. Commun. ACM **60**, 31–34 (2017)
D. Horsman, V. Kendon, S. Stepney, Abstraction/representation theory and the natural science of computation, in *Physical Perspectives on Computation, Computational Perspectives on Physics*, ed. by M.E. Cuffaro, S.C. Fletcher (Cambridge University Press, Cambridge, 2018), pp. 127–149
H. Jaeger, The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. GMD Technical Report **148**, 34. German National Research Center for Information Technology, Bonn, Germany (2001a)
H. Jaeger, Short term memory in echo state networks. GMD-Forschungszentrum Informationstechnik (2001b)

F. Jiang, H. Berry, M. Schoenauer, Supervised and evolutionary learning of echo state networks, in *Parallel Problem Solving from Nature (PPSN X)* (Springer, 2008), pp. 215–224
V. Kendon, A. Sebald, S. Stepney, M. Bechmann, P. Hines, R.C. Wagner, Heterotic computing, in *International Conference on Unconventional Computation (UCNC 2011)* (Springer, 2011), pp. 113–124
V. Kendon, A. Sebald, S. Stepney, Heterotic computing: past, present and future. Philos. Trans. R. Soc. A: Math. Phys. Eng. Sci. **373**(2046), 20140225 (2015)
Z. Konkoli, S. Nichele, M. Dale, S. Stepney, Reservoir computing with computational matter, in [68] (2018), pp. 269–293
A.F. Krause, V. Dürr, B. Bläsing, T. Schack, Multiobjective optimization of echo state networks for multiple motor pattern learning, in *18th IEEE Workshop on Nonlinear Dynamics of Electronic Systems (NDES 2010)* (IEEE, 2010)
R. Legenstein, W. Maass, Edge of chaos and prediction of computational performance for neural circuit models. Neural Netw. **20**(3), 323–334 (2007)
Lehman, K.O. Stanley, Exploiting open-endedness to solve problems through the search for novelty, in *ALife XI* (2008), pp 329–336
J. Lehman, K.O. Stanley, Evolving a diversity of virtual creatures through novelty search and local competition, in *Proceedings of the 13th annual conference on Genetic and Evolutionary Computation (GECCO 2011)* (ACM, 2011), pp. 211–218
S. Lloyd, Ultimate physical limits to computation. Nature **406**(6799), 1047 (2000)
W. Maass, T. Natschläger, H. Markram, Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput. **14**(11), 2531–2560 (2002)
M. Massey, A. Kotsialos, D. Volpati, E. Vissol-Gaudin, C. Pearson, L. Bowen, B. Obara, D. Zeze, C. Groves, M. Petty, Evolution of electronic circuits using carbon nanotube composites. Sci. Rep. **6**, 32197 (2016)
F. Matzner, Neuroevolution on the edge of chaos, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017)* (ACM, 2017), pp. 465–472
J.F. Miller, K. Downing, Evolution in materio: Looking beyond the silicon box, in *NASA/DoD Conference on Evolvable Hardware 2002* (IEEE, 2002), pp. 167–176
J.W. Mills, Polymer processors. Technical report TR580, Department of Computer Science, University of Indiana (1995)
J.W. Mills, The nature of the extended analog computer. Phys. D **237**(9), 1235–1256 (2008)
M. Mohid, J.F. Miller, S.L. Harding, G. Tufte, M.K. Massey, M.C. Petty, Evolution-in-materio: solving computational problems using carbon nanotube-polymer composites. Soft. Comput. **20**(8), 3007–3022 (2016)
J.B. Mouret, J. Clune, Illuminating search spaces by mapping elites (2015), arXiv:1504.04909
J.B. Mouret, S. Doncieux, Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity, in *IEEE Congress on Evolutionary Computation (CEC 2009)* (IEEE, 2009), pp. 1161–1168
J.B. Mouret, S. Doncieux, Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol. Comput. **20**(1), 91–133 (2012)
Y. Paquot, F. Duport, A. Smerieri, J. Dambre, B. Schrauwen, M. Haelterman, S. Massar, Optoelectronic reservoir computing. Sci. Rep. **2**, 287 (2012)
G. Pask, Physical analogues to the growth of a concept, in *Mechanisation of Thought Processes, National Physical Laboratory Symposium 10, HMSO*, vol. II (1959), pp. 877–922
J.E. Pearson, Complex patterns in a simple system. Science **261**(5118), 189–192 (1993)
J.K. Pugh, L.B. Soros, K.O. Stanley, Quality diversity: a new frontier for evolutionary computation. Front. Robot. AI **3**, 40 (2016)
J. Qiao, F. Li, H. Han, W. Li, Growing echo-state network with multiple subreservoirs. IEEE Trans. Neural Netw. Learn. Syst. **28**(2), 391–404 (2017)
A. Rodan, P. Tiňo, Simple deterministically constructed recurrent neural networks, in *International Conference on Intelligent Data Engineering and Automated Learning* (Springer, 2010), pp. 267–274

A. Rodan, P. Tino, Minimum complexity echo state network. IEEE Trans. Neural Netw. **22**(1), 131–144 (2011)
L.A. Rubel, The extended analog computer. Adv. Appl. Math. **14**(1), 39–50 (1993)
A.T. Sergio, T.B. Ludermir, PSO for reservoir computing optimization, in *International Conference on Artificial Neural Networks* (Springer, 2012), pp. 685–692
S. Stepney, Embodiment, in *In Silico Immunology*, ed. by D. Flower, J. Timmis (Springer, 2007), pp. 265–288
S. Stepney, The neglected pillar of material computation. Phys. D **237**(9), 1157–1164 (2008)
S. Stepney, Co-designing the computational model and the computing substrate, in *International Conference on Unconventional Computation and Natural Computation (UCNC 2019)*. LNCS, vol. 11493 (Springer, 2019)
S. Stepney, S. Rasmussen, M. Amos (eds.), *Computational Matter* (Springer, 2018)
G. Tanaka, T. Yamane, J.B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, A. Hirose, Recent advances in physical reservoir computing: a review. Neural Netw. **115**, 100–123 (2019)
A. Thompson, An evolved circuit, intrinsic in silicon, entwined with physics, in *Evolvable Systems: From Biology to Hardware (ICES 1996)* (Springer, 1997), pp. 390–405
A.M. Turing, Intelligent machinery, in *Machine Intelligence*, vol. 5, ed. by B. Meltzer, D. Michie (Edinburgh University Press, 1969), pp. 3–23 (published after the author's death)
J. Yperman, T. Becker, Bayesian optimization of hyper-parameters in reservoir computing (2016), arXiv:1611.05193
S. Yuenyong, On the gradient-based sequential tuning of the echo state network reservoir parameters, in *Pacific Rim International Conference on Artificial Intelligence* (Springer, 2016), pp 651–660