NTNU
Norwegian University of
Science and Technology

Exploring Physical Reservoir Computing using Random Boolean Networks.

Aleksander Vognild Burkow

Master of Science in Computer Science
Submission date: June 2016
Supervisor: Gunnar Tufte, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Reservoir computing (RC), a relatively new approach to machine learning, utilizes untrained recurrent neural networks as a reservoir of dynamics to pre-process some temporal task, making it separable with a linear readout layer. Potentially any sparsely connected network containing feedback loops can be a reservoir. One such network is the random Boolean network (RBN), which has previously been used in a reservoir computing setting.

Reservoir computing can be performed with physical or simulated reservoirs. When using a physical reservoir for computation, as in evolution-in-materio, one is often restricted in how one can perturb and read out the underlying substrate. In this thesis, the following properties of RBN RC devices are investigated and related to physical reservoirs: How task difficulty affects required reservoir size, how much reservoir perturbation is optimal, the performance of reservoir subsets, and the relationship between an RBNs attractors and its performance as a reservoir.

Experiments confirm that the required reservoir size increases with the difficulty of the task at hand, with the largest factor being how many bits of input the reservoir is required to remember. Simulation of RBN RC systems can therefore aid in deciding the optimal size of physical reservoirs, given a bridge between the computational power of the reservoir and RBNs can be deduced. Optimal reservoir perturbation is found to lie at roughly 50% of the size of the reservoir for RBNs with $K = 3$. When using smaller slices of a reservoir for computation, lower amounts of total perturbation will be required as long as these perturbations are located within the same topological area. Results also show that subsets of larger reservoirs will perform at least as well as a separate reservoir of equal size. Any interference from the unused parts of the reservoir is either minimal or slightly positive. Finally, no relationship is found between the attractors of a RBN and its performance in a RBN RC system. It can therefore not be used for guiding the construction of accurate RBNs.

Sammendrag

Reservoir computing (RC) er en relativt ny teknikk innen maskinlæring. Et utrent rekurrent nevralt nettverk benyttes som et reservoar fyllt av dynamikk for å preprossesere et temporært problem, og dermed gjøre det separerbart med et lineært avlesingslag. Alle glissne nettverk med tilbakekoblinger kan potensielt benyttes som et reservoar. Et slikt nettverk det tilfeldige Boolske nettverket (RBN), som tidligere har blitt brukt i RC-systemer.

Reservoir computing kan benytte både fysiske og simulerte reservoar. Når et fysisk reservoar brukes for beregning, som i evolution-in-materio, har man ofte begrenset adgang til å påvirke samt lese ut verdier fra materialet. I denne avhandlingen vil følgende egenskaper ved RBN RC-systemer bli undersøkt og relatert til fysiske reservoarer: Hvordan kompleksiteten til oppgaven påvirker størrelsen på reservoaret, hvor mye påvirkning som er optimalt, hva ytelsen på subsett av større rerservoarer er, og hva forbindelsen mellom et RBNs tiltrekkere og dets ytelse som reservoar er.

Våre eksperimenter bekrefter at den nødvendige størrelsen på reservoaret øker i takt med vanskelighetsgraden på oppgaven som skal løses. Den største faktoren er hvor mange bits med input som må huskes. Simuleringer av RBN RC-systemer kan derfor hjelpe i å avgjøre den optimale størrelsen på et fysisk reservoar, gitt man finner en bro mellom målet for beregningskraft på det simulerte og fysiske reservoaret. Optimal reservoarpåvirkning er funnet til å ligge på rundt 50% av størrelsen på reservoaret (for RBN med konnektivitet 3). Når man bruker et subsett av et reservoar for beregning så trengs det mindre mengder med påvirkning, gitt at påvirkingen skjer i samme del av reservoaret som leses av. Våre resultater viser også at subsett av større reservoar yter like bra som separate reservoar av den størrelsen. En eventuell interferens mellom den ubrukte delen av reservoaret og den brukte er enten neglisjerbar eller svakt positiv. Det viste seg ingen sammenheng mellom tiltrekkerne til et RBN og dets ytelse i et RBN RC-system. Dette kan dermed ikke brukes for å guide konstruksjonen av nøyaktige RBN.

Acknowledgements

I would like to thank my advisor Professor Gunnar Tufte for sharing his knowledge with me, and guiding my research through this weird and awesome part of Computer Science.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1 INTRODUCTION

Reservoir computing (RC) is a category of machine learning that sprang out from the study of recurrent neural networks (RNNs). It utilizes untrained recurrent neural networks as a reservoir of dynamics to preprocess a time series problem, transforming it from a temporal to a spacial one in the reservoir, making it separable with a simple readout layer [17].

Reservoir computing can be used both with physical or simulated reservoirs. The original Echo State Networks [13] and Liquid State Machines [21] provided the basis for reservoir computing. As alternative abstractions to RNNs, both cellular automata (CA) [30] and random Boolean networks [27] were introduced and successfully used in reservoir computing frameworks. Random Boolean networks (RBNs) [10] are an useful abstraction over many physical phenomena, most famously used by Kauffman [15] as a model for the genetic regulatory network. RBNs used in reservoir computing systems are known as RBN RC systems. In the author's pre-thesis paper [3], findings about RBN RC systems from [27] were reproduced and the reusability of trained readout layers investigated. A neutrality in the state-space of possible RBNs was discovered.

Physical reservoirs take many shapes and forms. In the frequently cited 'Pattern recognition in a bucket' paper [9], the authors use a bucket of water to successfully perform speech recognition. In [8], a carbon nanotube based substrate is successfully used to evolve binary logic gates as well as stable cellular automata. These can then be used to build a CA-based computing paradigm over this unconventional hardware. As cellular automata are a special case of random Boolean networks, any general findings about RBN RC systems may be applicable to RC systems using CAs. The wider category of attempting to harness the power of unconventional physical devices for computation, usually aided by artificial evolution, is known as evolution-in-materio [20].

When using nontraditional physical devices for computation, as in evolution-in-materio, one is often restricted in how one can perturb and read out the underlying substrate. It might be prohibitively expensive or technologically infeasible to read the entire state of the reservoir. In [4] the authors use living rat neurons in a microelectrode array to control an airplane in a flight simulator. Only subsets of the computational substrate are used, and the neuronal connections even change over time. In addition, the microelectrode array presents a limited resolution view of the computational substrate. Questions include how the unused parts of the reservoir impact the parts used for computation, and how large the reservoir actually needs to be to solve the task at hand.

It has been shown that reservoir computing using random Boolean networks is a feasible approach to solving binary time-series problems, as well as being a potentially useful abstraction over physical reservoir computing devices. In this thesis, the properties of these RBN RC systems are further investigated, with the motivation of using the results as a template for instrumenting and performing computations on physical reservoir computing systems.

The following research questions will be answered in this thesis:

1. How small can a RBN RC system be while still solving its task at $\geq 98\%$ accuracy?

2. Does the optimal amount of reservoir perturbation depend on the task at hand?

3. Does one have to read the state of the entire reservoir to maintain task accuracy?

4. Is there a correlation between the topological characteristics of the RBN (it's number of attractors and their lengths) and its performance as a reservoir?

Questions two and three are of specific interest to the implementation of physical reservoirs and evolution-in-materio devices. What these approaches have in common is that physical substrates, with varying degrees of perturbation ability and insight into the internal state, are used for computation. Their couplings to theoretical frameworks such as reservoir computing motivate the exploration of the effects of limited perturbation and readout possibilities on reservoir performance.

1.1 Thesis Structure

The thesis is structured as follows: Chapter 2 provides background information on reservoir computing, random Boolean networks, evolution-in-materio, and using RBNs for reservoir computing. Chapter 3 describes the experimental setup and methodology used for performing experiments. Chapter 4 contains an in-depth descriptions of each experiment, the results obtained, and a discussion. Chapter 5 concludes the thesis and mentions further work.

CHAPTER

2 BACKGROUND

2.1 A Brief Introduction to Reservoir Computing

Recurrent Neural Networks (RNNs), as opposed to feed-forward neural networks, are notoriously time consuming and difficult to train [25]. This is due to feedback from the recurrent connections during the training process, allowing small topology changes to drastically change a network's position in the fitness landscape.

It was therefore proposed in both [13] as Echo State Networks (ESNs) and [21] as Liquid State Machines (LSMs) to separate the RNN into two parts, the untrained recurrent reservoir, and the trained readout layer. The LSM and ESN methods have been unified into the field of Reservoir Computing, now focusing on the separate training and evolution of the recurrent and readout parts [17]. Exiting applications of Reservoir Computing include speech and handwriting recognition, as well as controlling robotics [17].

A generalized model of a Reservoir Computing system is shown in figure 2.1. The main components are the reservoir and the readout layer. At each time step in the core model, the reservoir receives the current input signal as well as its previous state. The reservoir transforms the input, and passes it on to the readout layer. The readout layer frequently receives the input signal as well. The internal weights of the reservoir are usually randomized and left untrained, with the weights of the readout layer being adjusted by some learning algorithm. This can be linear or ridge regression for offline learning, or recursive least squares for online learning [25].

There are many extensions to the base model. The reservoir and readout layers can receive a constant bias: The readout's bias is used for regularizing the reservoir state in case the problem is ill-posed. This isn't needed when using a model like ridge regression,

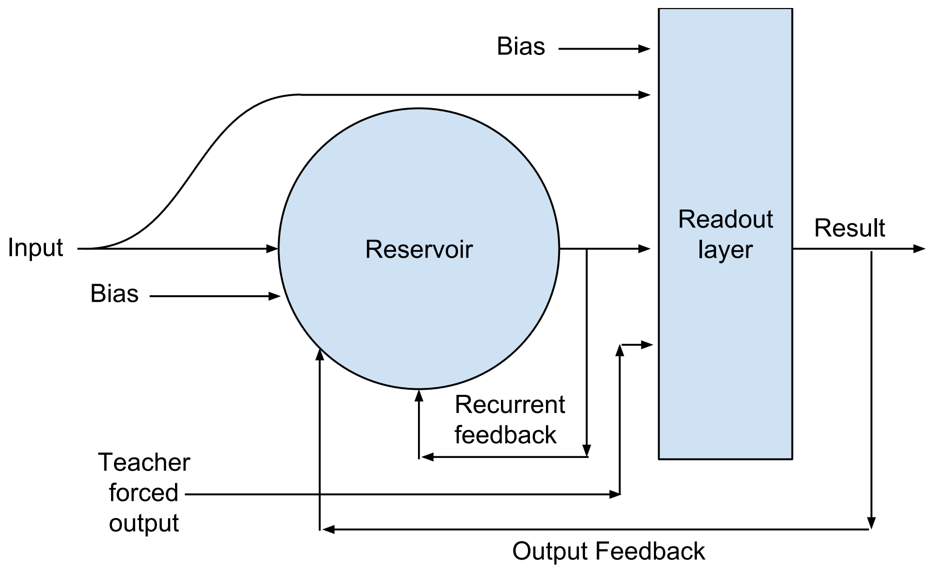


Figure 2.1: Schematic of a general Reservoir Computing system containing adjustable biases, feedback loops, reservoir and readout layers, which are described in section 2.1. Inspired by figures from [25] and [12].

which performs regularization internally [25]. The reservoir's bias is used for stabilizing models which feeds the readout layer back into the reservoir, which may be needed if the problem entails producing oscillating output [12]. Teacher forcing, that is forcing the states of the readout layer to those expected by the trainer for the first n time steps, usually speed up the convergence of the learning method used, and may in some cases be required to at all achieve stability [14].

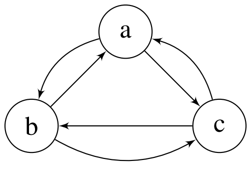
For a deeper dive into Reservoir Computing, consult papers [25], [17], and [12].

2.2 Alternatives to Traditional Reservoirs

Any complex dynamical system can in principle be used in a reservoir computing setting. What properties must these reservoirs have to be able to solve problems?

Complex networks similar to the sparsely connected ones used in ESN and LSM systems include cellular automata [29] and random Boolean networks [15]. Cellular automata are regular grids of cells containing some state, each cell connected to its neighbors in the grid. Cells then update in lockstep according to some shared transition table, creating a new generation. RBNs can be seen upon as an abstraction over CAs again, allowing for nonlocal neighbors and variable updating rules. This computational paradigm is known as Cellular Computing, and provides a potentially powerful alternative to classical computers, leveraging extreme parallelism, simple components and local state [26].

Both CAs and RBNs have successfully been used in reservoir computing systems [30] [27]. The RBN reservoir computing approach will be referred to as RBN RC. Both models



(a) RBN topology

Ancestor states		New state
0	0	0
0	1	1
1	0	0
1	1	1

(b) Transition table for node a

Figure 2.2: An example homogeneous RBN with $N = 3, K = 2, P = 0.5$.

are simple, and can be implemented in software, hardware such as FPGAs, and in materio [20] [8].

2.3 A Brief Introduction to Boolean Networks

Random Boolean networks were originally developed as a model of gene regulatory networks [15], the complex system that regulates how genes in multicellular organisms interact with each other. The model requires no assumptions about the inner workings of the actual nodes, which allows it to model phenomena where the exact internal workings of the system may be unknown.

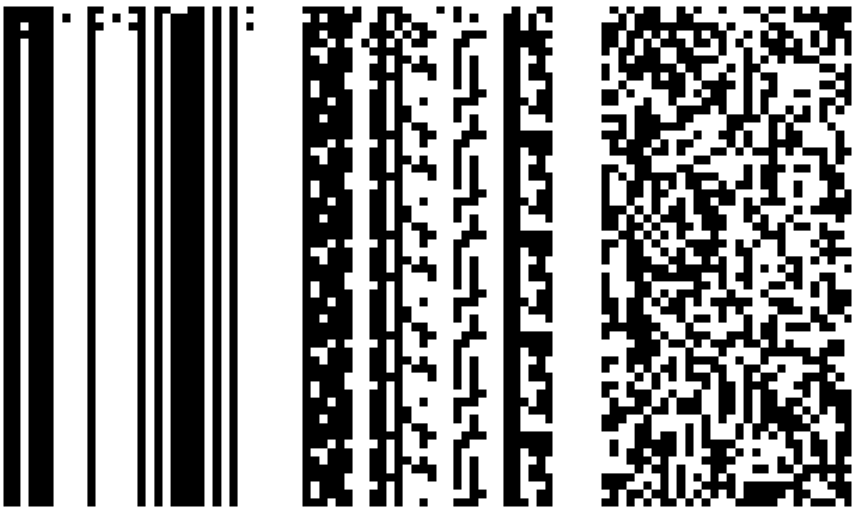
The simplification of a system to a Boolean model doesn't pose a problem, as any multivalued network can be transformed to a corresponding binary one.

A RBN is usually described by its number of nodes N and the in-degree K of the nodes, that is, how many nodes each node depends on (also known as its ancestors). RBNs can have both homogeneous and heterogeneous in-degrees. In heterogeneous networks, one usually describes the average connectivity $\langle K \rangle$ instead.

Each node can have a state of zero or one. The next state of the node is solely determined by the current combination of states of its ancestors. Each combination leads to a new state of zero or one, with the probability given by a binomial distribution usually having $\langle P \rangle = 0.5$. Figure 2.2 visualizes a homogeneous RBN with $N = 3, K = 2, P = 0.5$.

In the simplest RBN updating scheme, all nodes update in lockstep. This is known as the Classical RBN updating scheme (CRBN). The states of the RBN at the next time step $t + 1$ therefore only depend on the states at the previous time step t. As the number of RBN states is finite (2^{n_nodes}), the system will eventually revisit a previously visited state. This set of repeating states is known as an *attractor*, and a deterministic system cannot escape from it. If the attractor consists of a single state it is known as a point attractor, otherwise a cycle attractor. The set of states leading towards an attractor is known as its *basin of attraction*. A cycle attractor can be observed in figure 2.3b, while a point attractor is observed in figure 2.3a.

A criticism of the classical model is that gene regulation networks are updating continuously, as opposed to in lockstep [10]. There are therefore a number of alternate updating schemes which can be categorized by whether they are deterministic or nondeterministic, as well as synchronous and asynchronous.



(a) Ordered phase, K=1 **(b)** Critical phase, K=2 **(c)** Chaotic phase, K=3

Figure 2.3: Trajectories through state-space for RBNs with $N = 30, K = [1, 2, 3]$, visualizing the different phases. Time flows downwards the lattice, while RBN states are shown along the X-axis. with the network states plotted horizontally, and time flowing downwards. Images created with the developed RBN-simulator.

The dynamics of a RBN can be categorized as being in either the ordered, critical, or chaotic phase. These phases can be identified by how large a part of the network state is able to change over time, whether similar states tend to converge or diverge over time, and the networks resistance to perturbations (outside changes to the network).

One way to obtain these phases analytically is by comparing the resulting states of two identical RBNs where one is subject to some perturbation [10]. For visual identification, we plot the states of the RBN in a square lattice, with the network states plotted horizontally, and time flowing downwards. A node is drawn as white if its state is one, black otherwise. The phases are visualized in figure 2.3.

In general, RBNs in the critical phase are the most interesting. These are seemingly able to support information transmission, storage and modification, all capacities required for computation [16]. Critical systems are found on the edge of chaos, on the phase transition between ordered and chaotic networks [10]. For RBNs with $\langle p \rangle = 0.5$, critical dynamics are usually found at $\langle K \rangle = 2$ [10], although one could still find networks with such dynamics for different values of $\langle K \rangle$.

A thorough introduction to the field of RBNs is available in [10].

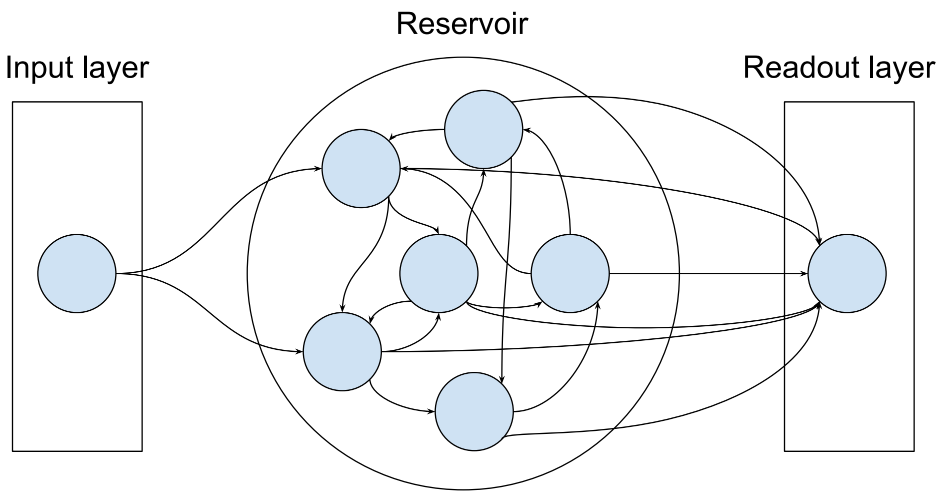


Figure 2.4: An example RBN RC system with $I = 1, IC = 2, K = 2, N = 6$ with the entire reservoir sate used for regression. The reservoir transforms the problem from a temporal one to a multidimensional spatial one. The readout layer the performs some kind of learning on the reservoir states against the expected output for the current task.

2.4 RBN Reservoir Computing Systems

How does one adapt a RBN for use as a reservoir in a RBN RC device? RBNs aren't usually designed to take external input. We do however, have the concept of perturbation, the external flipping of bits in the network's state, transition tables or edges. This can be utilized to create RBNs that take input, by continuously perturbing the RBN nodes by the bits of the input sequence.

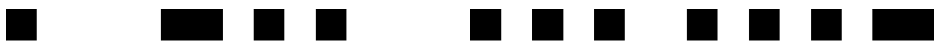
Questions that follow are how many bits should the network consume at a time, how many of the network nodes should be perturbed by the input at each time step, and what dynamics must such a reservoir have to allow for the computation of interesting problems?

2.4.1 A Working System

In [27] functioning RBN RC systems are created and analyzed. These RBN RC systems have heterogeneous connectivity, consume one bit of input at each time step ($I = 1$), perturbing IC of the N nodes in the process. The readout layer can be any node performing some kind of regression of the reservoir state against expected output for the current task, e.g. linear regression. Such a setup is shown in figure 2.4.

2.4.2 Tasks

To measure the real-life performance and accuracy of the RBN RC system, two tasks are introduced: Temporal density and temporal parity [27]. Both require the reservoir to be able to retain information for a sliding window of size n, offset by some value t, back


(a) Input

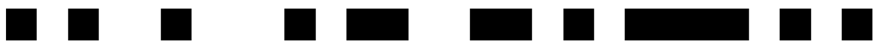

(b) Correct output

Figure 2.5: The first 30 elements of a temporal parity task with $[n = 3, t = 0]$. A one is visualized as white, while a zero is black. We see that correct output at time i is equal to there being an odd number of 1s in inputs $[i, i-1, i-2]$

through the input stream. The temporal parity task requires us to determine if there were an odd number of ones in the sliding window, the temporal density task to determine whether there were a majority of ones. The Former is visualized in figure 2.5.

Both tasks will be used to benchmark the reservoirs created later in this paper.

2.4.3 Computational Capability

For a RBN-reservoir to perform well at computational tasks, it must be able to both forget past perturbations and keep two input streams that have begun converging separated [2].

These two properties are coined *fading memory* and *separation property*, and can be measured [27] as follows.

Create two equal input streams #1 and #2 of length T. If measuring *fading memory*, flip the first bit in stream #2. If measuring *separation property*, flip all bits up to bit $T-t$ in stream #2 (t being the required depth of separation). For both input streams, reset reservoir state, perturb the reservoir with the input stream, and store the final state. The score of the measure is then defined as the normalized hamming distance between the resulting states. The computational capability Δ of a RBN-reservoir is then defined as

$$\Delta_{Tt} = separation\_property_{Tt} - fading\_memory_{T} \tag{2.1}$$

Analyzing different RBN-reservoirs with this metric [27], a high Δ is found to correlate with critical connectivity ($\langle K \rangle = 2$). For all RBN-reservoirs, Δ drops when increasing the required separation t, and is maximized when one doesn't have to remember anything at all ($t = 0$).

2.4.4 Optimal Perturbation

It is found that the optimal amount of reservoir perturbation, adjustable by the number of connections between the input layer and the reservoir, depends on both the task size, how many steps in time are required to be remembered, and the dynamics of the reservoir. *Chaotic reservoirs* require few input connections to be able to properly spread information, but perform poorly on larger tasks due to past perturbations still floating around the reservoir. *Ordered reservoirs* quickly forget past perturbations, allowing some success for larger tasks, but their inability to remember past perturbations renders them useless for many tasks. *Critical reservoirs* require connectivity somewhere in the middle. Able to forget as well as remember, they perform accurately independent of task size.

2.5 Evolution in Materio and Physical Reservoirs

The field of evolution-in-materio [20] concerns the harnessing of unconventional and complex physical materials, using them to perform computation. The process is usually aided by Artificial Evolution for finding reasonable input-substrate-output mappings. The difference from classical computation is that the computation is moved from an abstraction over the hardware (e.g. a modern CPU), and into the material itself. This material can be anything from grown dendritic iron wires which discriminate sound [22] to carbon nanotubes used for evolving CA rules [8]. Success within this field may result in more efficient use of materials, and higher computational densities.

In [4], living rat neurons are put in a microelectrode array (MEA) and successfully used to control an airplane in a flight simulator. Two of sixty electrodes were chosen to control the pitch and roll of the airplane. The internal 'weights' of the rat brain were trained by administering a series of electrical impulses to the same electrodes which then increase or decrease the action potential of the region, correcting behavior. The internal state of the system is unknown, with the MEA only exposing a limited resolution subset of the entire system state (around 60 electrodes).

In [8], the computational power of carbon nanotubes, instrumented in a microelectrode array, is used to evolve the state transition functions of all elementary Cellular Automata. Artificial Evolution is used to find the correct translation of input/output values to their respective electrodes. The internal state of the carbon nanotubes are again completely opaque to the higher computational layers. In addition, the number of electrodes present a limited resolution view of the substrate.

In the ever-so-cited 'Pattern recognition in a bucket' paper [9], the Liquid State Machine metaphor is taken literally and a bucket of water is used as a physical reservoir. The bucket of water receives input via rotors mounted on the edge of the bucket. Input is then transformed from a temporal into a spatial, with a great deal of nonlinearity introduced by the bucket. Finally, the reservoir state is read out by taking pictures of the wave-patterns, using a simple perceptron to classify the resulting state. This setup correctly separates the xor problem as well as being able to classify the words 'one' and 'zero' when yelled into the bucket.

What these approaches have in common is that physical substrates, with varying degrees of perturbation ability and insight into the internal state, are used for computation. Their couplings to theoretical frameworks such as Reservoir Computing motivate the exploration of the effects of limited perturbation and readout possibilities on reservoir performance.

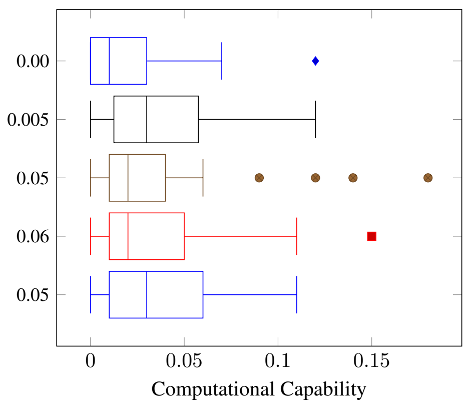
2.6 Validating RBNs for Reservoir Computing

In the authors pre-thesis paper [3], the dynamics, performance, and viability of RBN RC systems was investigated. A functioning RBN RC system was implemented, and its results validated against and found in accordance with those from [27]. The simulation software from the pre-thesis work has been greatly extended for use in this thesis.

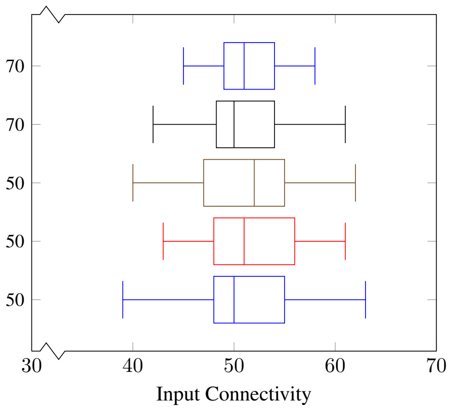
A positive correlation between the computational capability (section 2.4.3) of a reservoir and its actual performance is found. The optimal connectivity for homogeneous reservoirs is found to be $K = 3$ as opposed to $\langle K \rangle = 2$ for heterogeneous reservoirs [27]. Finally, the required input connectivity is found to rise with the presence of chaotic dynamics in the reservoir. The figures backing this conclusion are shown in 2.7 and 2.8.

Finally, a one-to-many mapping between the readout layer of an already-trained RBN RC system and different RBN RC is found, with there being a seemingly large set of interchangeable reservoirs for each readout layer. This neutrality in the space of possible RBNs make the potential use of a smaller generative genome for evolving RBN RC systems interesting. Even though such a genome will hit fewer points in the RBN fitness landscape than a fixed genome, a large amount of these points will still be usable for each instance of a working readout layer. There doesn't seem to be a tight correlation between the properties of the RBN from the original RBN RC system, and the equivalent RBNs found through artificial evolution. In fact, the computational capabilities of figure 2.6a and the connectivity distributions of figure 2.6b seem much more representative of the general RBN population, as shown in figure 2.7c. This indicates that while there are many compatible reservoirs for a given readout layer, the distribution of these compatible reservoirs are likely the same as the distribution of reservoirs with the same connectivity in general.

The parameters for the temporal parity task used for all experiments is shown in table 1 of appendix A.

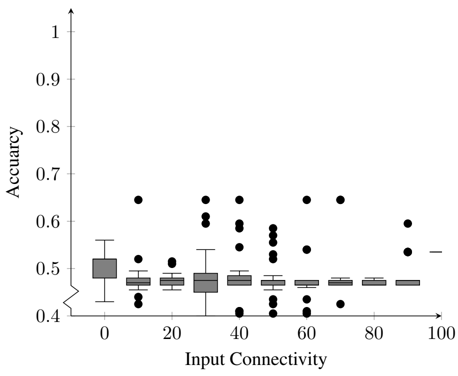


(a) Evolved RBN input connectivity

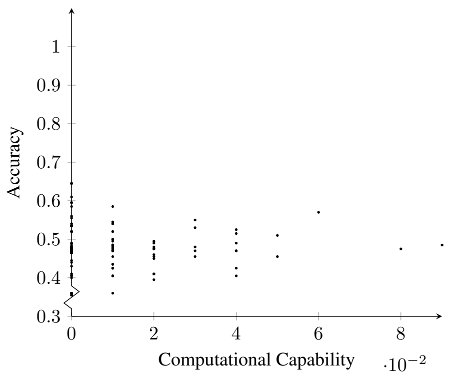


(b) Evolved RBN Computational Capability

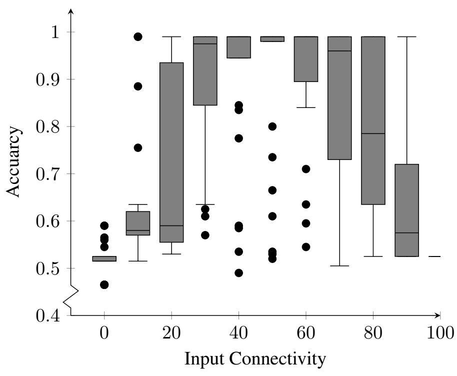
Figure 2.6: Computational Capability and Input Connectivity distributions for RBNs evolved from a fixed readout layer. The Y-axis shows the CC and IC values for the readout layers original RBN.



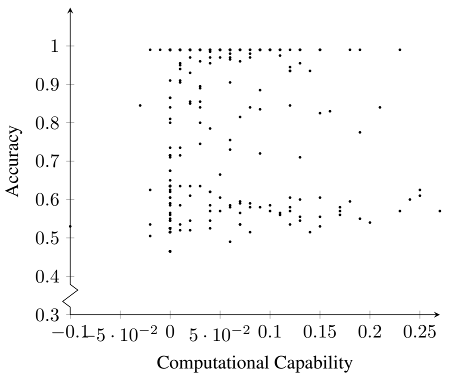
(a) K=1



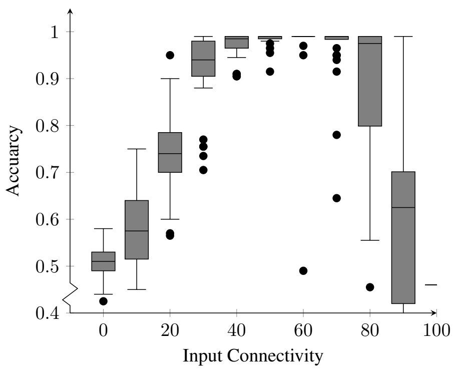
(b) K=1



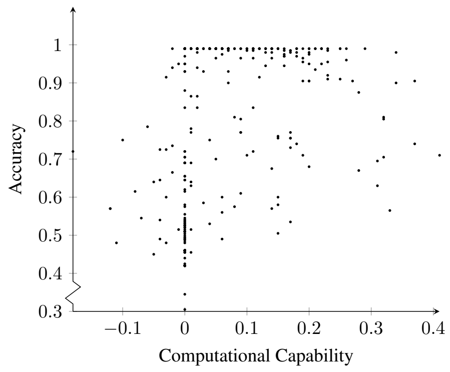
(c) K=2



(d) K=2

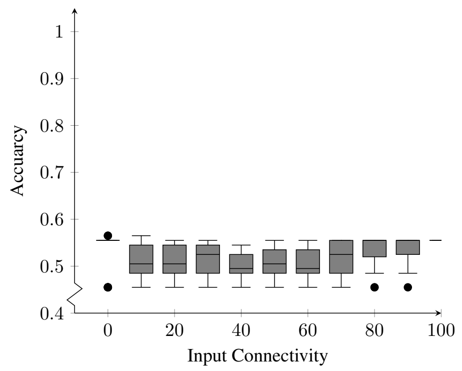


(e) K=3

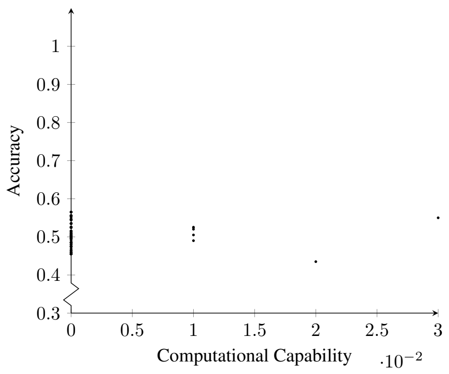


(f) K=3

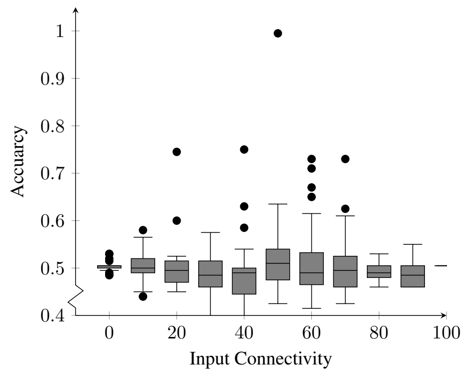
Figure 2.7: Plots for temporal parity 3. The left column shows the accuracies of the sampled RBNs against their input connectivity for K=1–3 respectively. The figures in the right column plot the accuracies of the figures to the left against their computational capabilities calculated for $T = 100, t = 3$.



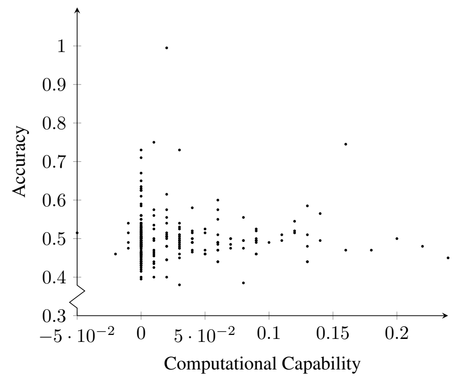
(a) K=1



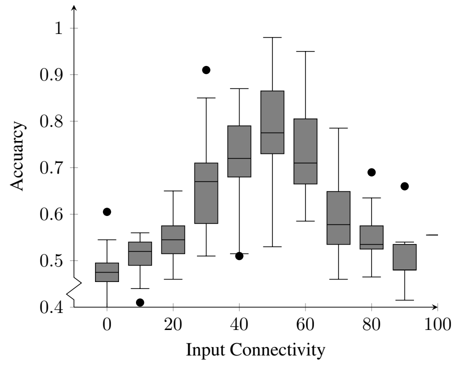
(b) K=1



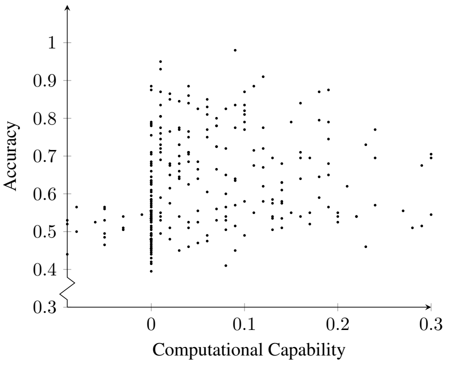
(c) K=2



(d) K=2



(e) K=3



(f) K=3

Figure 2.8: Plots for temporal parity 5. The left column shows the accuracies of the sampled RBNs against their input connectivity for K=1–3 respectively. The figures in the right column plot the accuracies of the figures to the left against their computational capabilities calculated for $T = 100, t = 3$. Note that the accuracies are much lower in general, with only $K = 3$ performing adequately.

CHAPTER

3 METHODOLOGY

A series of experiments will be performed to investigate the following research questions. They will shed some light on attributes of RBN RC systems that might be relevant to the creation and instrumentation of physical reservoirs.

1. How small can a RBN RC system be while still solving its task at $\geq 98\%$ accuracy?
2. Does the optimal amount of reservoir perturbation depend on the task at hand?
3. Does one have to read the state of the entire reservoir to maintain task accuracy?
4. Is there a correlation between the topological characteristics of the RBN (it's number of attractors, attractor length) and its performance as a reservoir?

3.1 Experimental Setup

The experimental setup used in this thesis is a further developed version of the RBN RC system used in [3]. It has been extended too support variable output connectivity as well as input connectivity, and is visualized in figure 3.1. The software is written in the Python programming language, and is available online on GitHub [18]. It contains a fully functioning RBN RC system, the classification tasks from section 2.4.2, RBN analytics tools, and a set of procedures for generating distribution statistics from experiment templates. The RBN RC system's readout layer uses scikit-learn's lightweight ridge regression layer [23]. Statistical and numerical operations are supplied by the NumPy library [28].

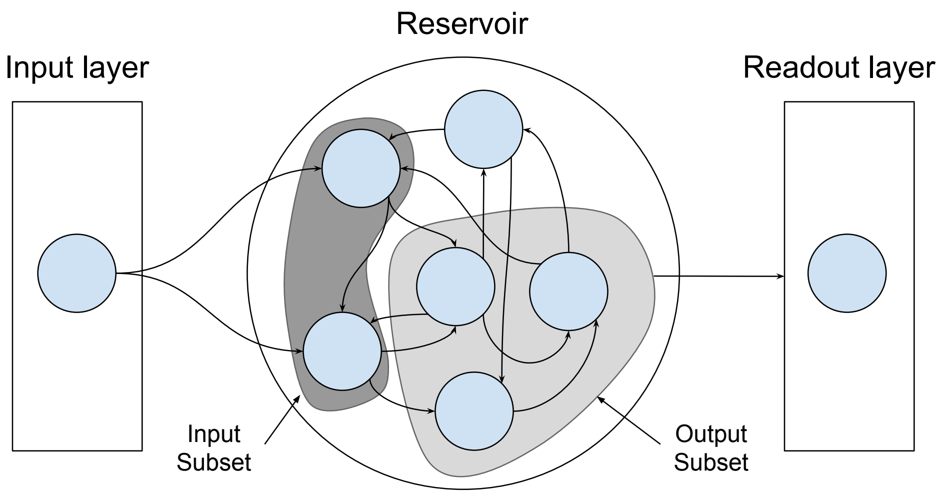


Figure 3.1: Illustration of the RBN RC system used in this thesis. RBN RC parameters are Input Nodes = 1, Input Connectivity IC = 2, Output Connectivity OC = 3, Number of nodes N = 6. The Input Connectivity and Output Connectivity is adjustable, and is shown with a shaded background.

3.1.1 RBN Parameters and Sample Sizes

As the number of different RBNs per N-K combination is oppressively large $(\frac{2^{2^K} N!}{(N-K)!})^N$ [10], the reservoir size, input, and output connectivity experiments will sample 50 RBNs for each experiment parameter combination. This sample size was regarded adequate as a sample size of 30 was shown to give satisfactory results for similar experiments in the pre-thesis work [3].

For the attractor analysis, the number of samples is increased to 500 per experiment parameter combination. This larger value is chosen as the other experiments have at least 10 parameter combinations, resulting in at least 500 samples.

Note that all our RBNs are of homogeneous connectivity, opposed to the heterogeneous connectivity used in [27]. This changes the optimal RBN connectivity for reservoir computing from $\langle K \rangle = 2$ to $K = 3$, as shown in section 2.6.

The RBN parameters shared for all experiments are shown in table 3.1.

Table 3.1: Common RBN parameters

Parameter	**Configuration**
Connectivity K	3
Bias p	0.5
Operation mode	CRBN (Classical synchronous RBN)

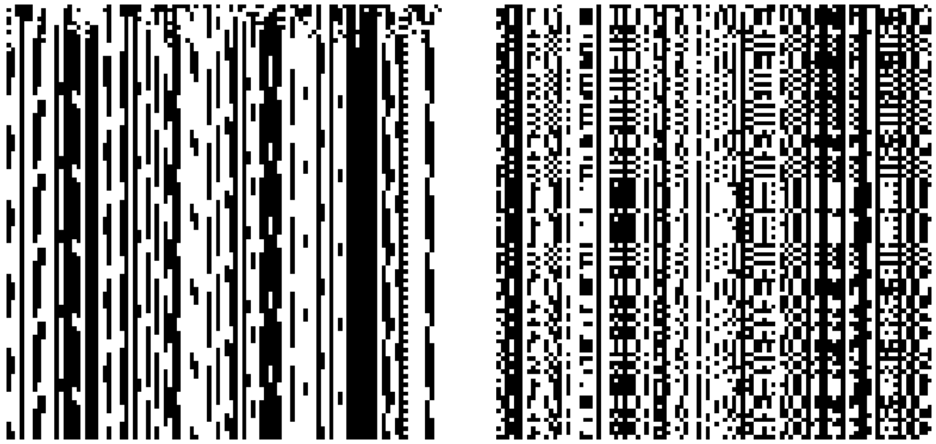


Figure 3.2: The same RBN ($N = 100, K = 2, P = 0.5, IC = 50$) shown both perturbed and unperturbed. The Boolean states of the RBN are plotted along the X-axis, with time flowing downwards.

3.1.2 Testing

To verify that RBN simulation is working, a RBN is created randomly, initial state set to all zeros, and ran. The results are visualized in Figure 3.2a. We see that the RBN exhibits stable dynamics, and enters into an attractor around $t = 15$. In Figure 3.2b we continuously perturb the RBN with the input stream from the temporal parity task visualized in Figure 2.5. In the perturbed case, the state trajectory is continuously changed, preventing the RBN from settling into an attractor. Interestingly enough, there seems to be a visual similarity between the two cases. Such a pattern is sure to disappear with a RBN in the chaotic phase.

This erratic pattern of state transitions is then fed into the readout layer, which is then tasked with finding a linear combination of the RBN states that results in the expected output for the given task.

3.1.3 Training

To train the RBN RC system we require large training datasets, as well as different, smaller datasets for testing the trained system. We will use the datasets described in section 2.4.

We then either create a new RBN (initialize it randomly), or load a previously created RBN from disk. For each bit of input in each dataset, we perturb the input-connected nodes in the RBN. After each perturbation, the RBN is ran synchronously (CRBN mode) for one time step. The resulting RBN states are collected, and after the entire dataset is processed, forwarded to the readout layer.

To find a suitable mapping from the set of reservoir states and the correct input classification, ridge regression [11] is used. This version of least squares regression is more accurate when faced with input collinearity, as well as always being at least as accurate as ordinary least squares. This process is repeated for all the datasets, and the final regression parameters are chosen as a combination of the parameters obtained for each individual dataset. Finally we measure the normalized accuracy of the trained reservoir on the test dataset, defined as the following:

$$Accuracy = 1 - \frac{sum(actual\_output \neq expected\_output)}{len(correct\_output)} \tag{3.1}$$

3.1.4 Computing the Attractors of a RBN

The number and length of attractors can vary greatly across different RBNs, but is largely determined by the connectivity of the system [10]. There are a number of ways to estimate the number of attractors and their properties of a specific RBN. They generally fall somewhere on the spectrum between exact enumeration (exhaustively searching for attractors from all initial states) and numerical sampling (searching from a subset only). Numerical sampling is biased for $K \neq 2$ [1], and cannot be used for our $K = 3$ RBNs. A brute-force exhaustive search quickly becomes infeasible, as the number of RBN states is exponential in the number of nodes.

The authors home-cooked exhaustive searcher times out for RBNs with more than 15 nodes. A SAT-Based algorithm for finding attractors in CRBNs, which was introduced in [7], is therefore used. It allows for an increase from 15 to 26 nodes, which should aid in finding more meaningful results. Its source code is available online at [6].

CHAPTER

4 EXPERIMENTS

4.1 Minimum Required Reservoir Size and Optimal Input Connectivity

4.1.1 Description

Section 2.6 and [3] show that almost all RBN RC systems with $N = 100, K = \{2, 3\}$ are able to solve the temporal parity 3 task. For temporal parity 5, reservoirs with $N = 100, K = 3$ are barely able to solve the task, with their $K = \{1, 2\}$ brethren being unable to classify the task correctly.

To find the minimum required reservoir size, as well as how it changes with the temporal memory requirements and complexity of the task at hand, we create a number of reservoirs with parameters from 4.2. The four tasks used to benchmark the generated reservoirs are temporal parity 3 and 5 (the number being the window size), as well as temporal density 3 and 5. These will sometimes be referred to as TP3, TP5, TD3, and TD5.

As the temporal density task is computationally less expensive than temporal parity [27], one can expect a smaller required reservoir size than for temporal parity. In addition, there should be a 'significant' bump in required reservoir size for larger task windows, as observed in section 2.6 and [3]. The expected relationship between the difficulties of the four tasks therefore becomes $TD3 \leq TP3 \leq TD5 \leq TP5$.

To find which input connectivity gives the greatest population accuracy, one iterates over all input connectivities, generating accuracy distributions for each one. In section 2.6, the optimal input connectivity is empirically observed to lie around $0.5 \cdot n\_nodes$.

Table 4.1: Task parameters. The tasks are explained in detail in chapter 2.4.2

Parameter	**Configuration**
Task type	Temporal parity and temporal density
Training dataset length	4 000
Test dataset length	200
N (window size)	3 and 5
t (offset)	0

Table 4.2: Reservoir parameters for optimal input connectivity

Parameter	**Configuration**
Nodes	10 to (100, 140) for TP, 5 to (35, 65) for TD
Node step size	10 for TP, 5 for TD
Connectivity	3
Input connectivity	[0..n_nodes], step size = 5
Output connectivity	n_nodes
Sample size	50

The chaotic reservoirs with $K = 3$ have a slight skew to the right of 0.5. In addition, the temporal density task might have different input connectivity requirements than the temporal parity task.

The resulting accuracy distributions will be plotted as box plots (as in section 2.6). This should allow for visual identification of the minimum required reservoir sizes, as well as the optimal input connectivities.

4.1.2 Results

We define a 'Task accuracy threshold' as the smallest reservoir size where at least two reservoirs have the required accuracy on the task. These are presented in figure 4.1 and table 4.3. The 90% accuracy threshold is also included in table 4.3 as it appears quite a bit earlier than the 98% threshold for tasks with a window size of 5.

Table 4.3: Accuracy thresholds for all four tasks.

	TP3	**TP5**	**TD3**	**TD5**
90% accuracy threshold	15	70	5	30
98% accuracy threshold	20	90	10	55

The number of input connectivity plots resulting from the reservoir (table 4.2) and task parameter (table 4.1) combinations is quite large. Therefore only reservoir sizes of $N = [10...30, 80...100]$ from the accuracy distributions on the temporal parity 3 task is shown here (figure 4.2). There is a slight skew to each side of $0.5 \cdot n\ _nodes$ dependent on whether the task chosen is temporal parity or temporal density. This is observable in figure 4.2 for TP3 and appendix B figures 1 through 7 for the remaining tasks.

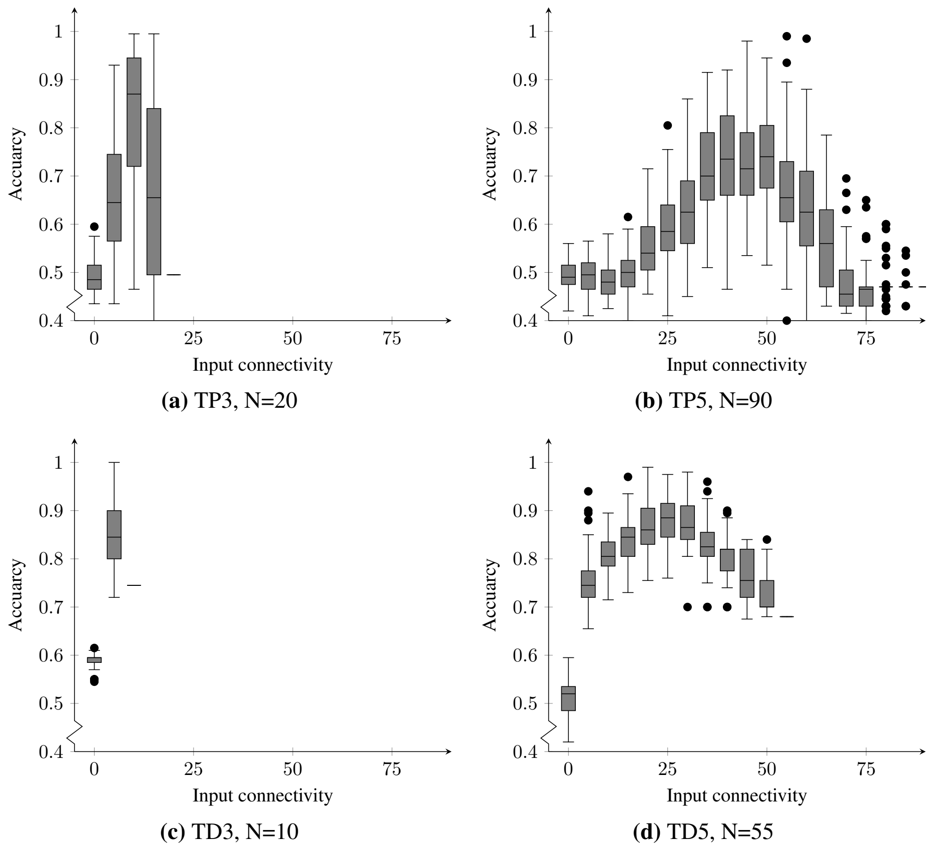


Figure 4.1: Accuracy plots for the required reservoir sizes to reach the 98% accuracy threshold for each of the four tasks: TP3 (Figure 4.1a), TP5 (Figure 4.1b), TD3 (Figure 4.1c) and TD5 (Figure 4.1d). The x-axis for all plots has been normalized to the largest reservoir size, $N = 90$.

We confirm this skew by calculating the optimal input connectivity for each task as follows:

$$optimal\_ic^{task} = average(max\_accuracy\_ic^{task}_{n\_nodes}/n\_nodes) \tag{4.1}$$

where $max\_accuracy\_ic^{task}_{n\_nodes}$ is the connectivity which gives the highest number of high-accuracy reservoirs for that task and reservoir size. The results are presented in table 4.4, with the chosen values of $max\_accuracy\_ic^{task}_{n\_nodes}$ presented in table 2 in appendix B.

Table 4.4: Optimal input connectivities as fraction of reservoir size.

	T=3	**T=5**
Temporal parity	0.528	0.489
Temporal density	0.439	0.443

4.1.3 Discussion

Required Reservoir Size

The suspected ordering of the task difficulties, $TD3 \leq TP3 \leq TD5 \leq TP5$, is confirmed in figure 4.1 and table 4.3. A reservoir of size 20 is large enough to correctly predict TP3, quite a bit lower than the 100 nodes used in section 2.6 (figure 2.7e). A reservoir of size 90 appears sufficient for TP5, rather close to the 100 nodes used in used in section 2.6 (figure 2.8e). The entire accuracy distribution doesn't increase significantly until 120 nodes, as seen in figure 4 in the appendix. A tiny reservoir of size 10 is sufficient for TD3, with a reservoir of size 5 achieving 90% accuracy! There is a corresponding ramp-up in required reservoir size to 55 to achieve 98% accuracy on TD5.

For the same window size, temporal parity requires a larger reservoir than temporal density, and the additional computational requirements of remembering five time steps into the past requires a significant ramp-up in reservoir size. The memory capacity of an Echo State Network trained on Independent and identically distributed data (as both our tasks are) is bounded by the size of the reservoir [12]. A randomly generated reservoir is statistically unlikely to be able to attain this bound, as additional constraints on the reservoir topology are required [12]. One such specially crafted RBN RC system is the one consisting solely of nodes passing their states forward to the next node each time step, only the first node being perturbed and all nodes read out. The memory bound for the RBN RC systems used here seem lower than their computationally more powerful ESN forefathers, which is corroborated by the significant increase in required reservoir size for tasks with window size 5. This isn't that surprising as the RBN RC system is a much simpler model of computation than the connected sigmoid nodes of the ESN RC system.

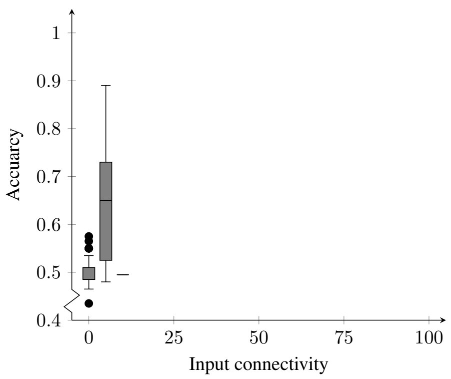
When using Reservoir Computing as an abstraction over a physical device (e.g. an evolution-in-materio device), empirical results from simulation can be used to aid in the suggestion of the required physical reservoir size. One would need a way to roughly quantify the computational power contained in the simulated RBN as well as that of the computational substrate, so that one can convert the results to meaningful physical sizes. This is the case for systems such as the cellular automata implemented in carbon nanotubes [8], as cellular automata are a special case of RBNs.

Optimal Input Connectivity

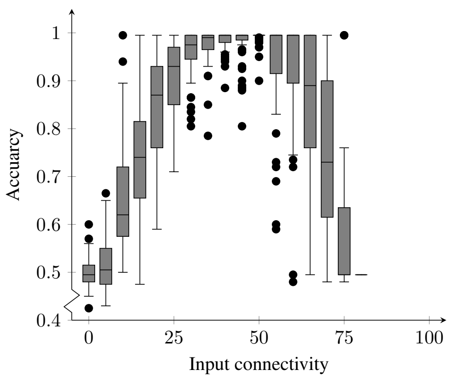
Temporal density had an optimal input connectivity of roughly 0.44, while temporal parity lay around roughly 0.50 (table 4.4). These values differed relatively little across tasks. This is to be expected, as the reservoir connectivity is the main factor in how large a reservoir perturbation is required [27], as discussed in section 2.4.4.

The net decrease in optimal IC from temporal parity 3 to 5 can be explained by the input needing to be remembered for longer in the network. A lower amount of perturbation results in less overriding of already-stored information in the reservoir. This conclusion is a bit hairy however, as there isn't a corresponding decrease in optimal IC for temporal density, which already lies at a low 0.44. These variations could also be explained by variance the optimal IC sampling process described in subsection 4.1.2.

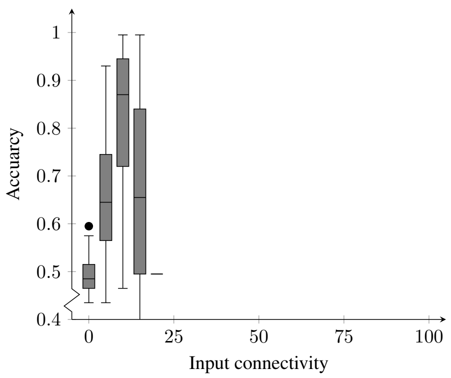
Finally, these findings can be used to assist in choosing the degree of perturbation in a real-life reservoir computing system, given a rough idea of what dynamical regime it may exist in (chaotic, critical, stable). When instrumenting the rat neurons for airplane automation in [4], only two of the sixty available microelectrodes of the MEA were used even though the computational substrate stretched under all of them. The same two microelectrodes were used for both perturbing the substrate and reading out its state. This indicates that the required amount of perturbation may depend on how large a part of the reservoir one actually uses, due to the spatial locality of computation in physical reservoirs. There might not be a need for additional perturbing of the neurons furthest away from the readout microelectrodes, as the effect might be minimal at the readout location.



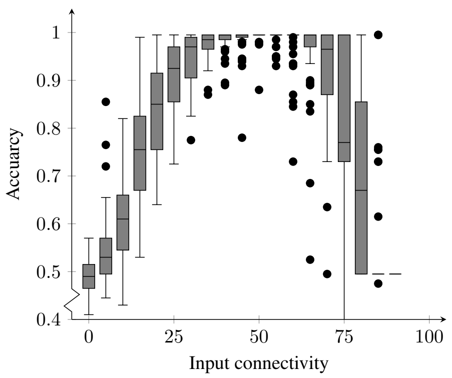
(a) N=10



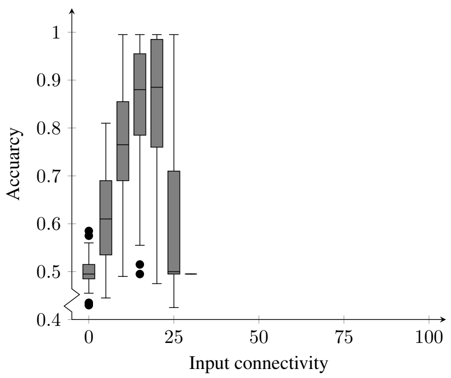
(b) N=80



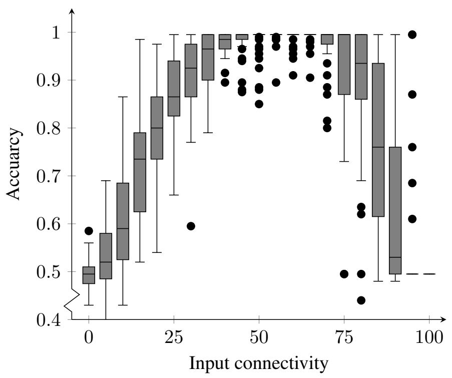
(c) N=20



(d) N=90



(e) N=30



(f) N=100

Figure 4.2: Plots of input connectivity against accuracy on TP3. Reservoir sizes [40..70] are omitted for brevity. Note that the optimal input connectivity tends slightly to the right of the middle for all reservoir sizes. The omitted plots are presented in figures 1 and 2 in appendix B.

4.2 Minimum Required Output Connectivity

4.2.1 Description

One isn't always able to fully instrument a computational substrate used for reservoir computing. While a microelectrode array gives coarse-grained access to smaller substrates than ever, one is still limited to a number of samples equal to the number of microelectrodes in the array. Other reasons can include instrumentation being prohibitively expensive, physically impractical or impossible, or quantum effects changing the material as one observes it. These situations result in less internal state and potentially less computational power being exposed to the user.

How does this restricted viewing of the reservoir state affect its predictive behavior? If there is an interference effect between the unobserved parts of the reservoir, does this effect increase the computational ability of the observed subset or decrease it? To test this hypothesis, chosen subsets of larger reservoirs will be compared to separate reservoirs of that exact size. This subset should perform better, equal, or worse than the independent reservoir, and therefore answer the hypothesis.

To reduce the search space, only temporal parity 3 and 5 will be used for these experiments. In addition, all reservoirs will have a fixed input connectivity of 50%, backed by the findings of optimal IC from the previous section (as summarized in table 4.3). Reservoir subsets will be chosen randomly from all available nodes, which only seems fitting given the random nature of the RBN and the lack of a meaningful physical projection. The reservoir parameters used for this experiment are shown in table 4.5, with the parameters for temporal parity staying the as for the previous experiments (table 4.1).

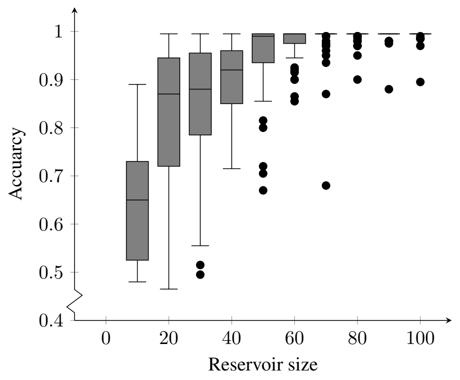
Table 4.5: Reservoir parameters for optimal output connectivity

Parameter	**Configuration**
Task	Temporal parity 3 and 5
Nodes	10 to (100, 140), step size=10
Connectivity	3
Input connectivity	$n\_nodes/2$
Subset size	[0..n_nodes], step size=10
Sample size	50

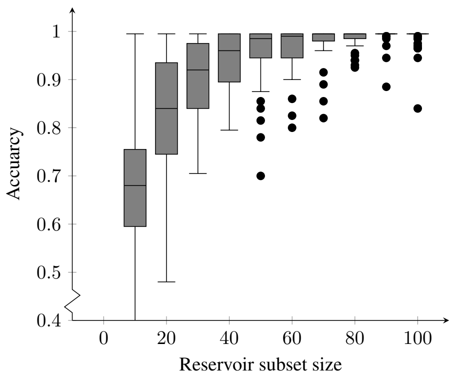
4.2.2 Results

For easier comparison of the performance of a reservoir subset against reservoirs of that exact size, plots aggregating the best accuracy distributions for each reservoir size are created. These are based on the accuracy distribution for the optimal input connectivity for that reservoir size, as shown in table 2 in appendix B. These are then compared to the subset-accuracy plots created from the parameters of table 4.5.

Figure 4.3 shows this comparison for task TP3, up to a reservoir size of 100. Figure 4.4 shows the same comparison for task TP5, up to a reservoir size of 140. The subset-accuracy plots for reservoir sizes ≤ 100 and ≤ 140 not presented here are shown in figures



(a) Best n-sized reservoir performance on TP3



(b) Reservoir subset performance on TP3

Figure 4.3: Accuracy plots for task TP3. Figure 4.3a shows the best performing n-sized reservoirs up to a size of 100. Figure 4.3b shows reservoir performance for subsets of reservoirs of size 100. All n-sized subsets perform virtually identical to their same-sized reservoirs.

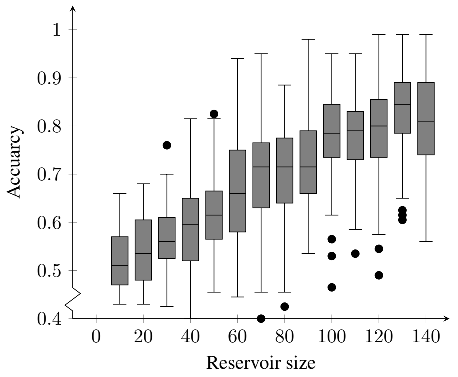
8 through 11 in appendix C.

4.2.3 Discussion

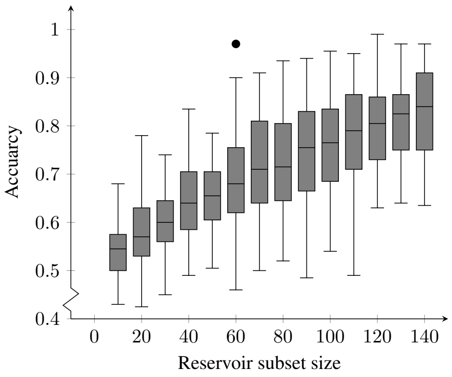
The resulting distributions from reservoir subsets and same-sized reservoirs are extremely similar, as seen in figures 4.3 and 4.4. The fact that reservoir subsets perform almost identically to the same-sized subsets indicates that any interference from the unused parts of the reservoir may be minimal. Additionally, as soon as the reservoir subset size is as large as the minimum required reservoir size, 20 for TP3 and 90-100 for TP5 (table 4.3), performance reaches a plateau with no need to further increase the size of the observed subset.

There seems to be a slight increase in population accuracy for small values of N (10, 20) however. An even larger sample size would probably be needed to decide whether this is randomness or positive interference from the unused parts of the reservoir. Selecting 10 nodes from 100 or 140, of which 50 and 70 respectively receive input each tick, might result in a heavy bias towards either side. Further, there is a possibility that slightly better subset performance could have been achieved by generating reservoirs for all ICs instead of using a flat IC of 50%, and then selecting the best one. Even though the average optimal IC is quite close to 50%, the backing data may lie slightly to either side, as seen in table 4.5.

Another concern is in what way, if any, a smaller set of reservoir states affects the model used for learning the mapping from reservoir states to classification. Using too many predictor variables in a linear model might result in overfitting the problem, while too few may result in low accuracies. The problem of redundant or useless variables is lessened if one uses an algorithm which includes regularization, such as ridge regression [11]. The regularization penalty rewards smaller coefficients which in turn leads to useless variables being eliminated almost completely, but they cannot be totally zeroed within



(a) Best n-sized reservoir performance on TP5



(b) Reservoir subset performance on TP5

Figure 4.4: Accuracy plots for task TP5. Figure 4.4a shows the best performing n-sized reservoirs up to a size of 140, Figure 4.4b shows reservoir performance for subsets of reservoirs of size 140. All n-sized subsets perform virtually identical to their same-sized reservoirs.

this model. Algorithmic regression variable subset selection which completely eliminates unneeded variables exists [19], but may not be useful for systems with a small readout-to-reservoir ratio. Such a system is the sixty electrodes stuck into a large soup of rat neurons [4]. Any redundancy in the neurons must surely be dominated by the nonlinearity each microelectrode aggregates from all nearby neurons, reducing the need for variable elimination.

In a continuously growing and expanding reservoir, such as the one consisting of rat neurons, the computational capabilities of the system would presumably grow monotonically with the increase in matter to the system. These results indicate that as long as one has instrumented a large enough part of the reservoir in the first place, one will not have to re-instrument it in the future (as long as the task doesn't change to a significantly more complex one requiring additional data points). The readout layer might have to be recalibrated, however. A regression model trained on a certain reservoir might not work as well on one with a slightly altered topology. In section 2.6 it is found that there exists a neutrality in the set of RBNs used for reservoir computing. That is, if one were to re-use a trained readout layer from a well-performing reservoir, there will be a number of different RBNs that perform accurately together with the re-used readout layer. It was not investigated whether these functionally equivalent reservoirs were structurally similar. If they are, one might not have to retrain the model every time the substrate evolves, as small changes to the topology might jump to a similarly well-performing RBN due to the inherent neutrality.

4.3 Analyzing Reservoir Dynamics

4.3.1 Description

When analyzing Random Boolean Networks one often does so in aggregate, looking at the population averages of the attributes of RBNs of a certain construction. This due to there being large variances in individual RBN dynamics [10]. One often looks at whether the average network is chaotic, stable, or critical, and the number of attractors, their size, their basins, and state-space transient times. The expected averages of these properties can then be obtained given a networks connectivity and size.

One indicator of reservoir performance used both in [27] and in the pre-thesis work [3] is Computational Capability, but what others are there? Is there a deeper link between a well-performing RBN used for computation and its topology and dynamical properties? What correlation, if any, is there between a RBNs number of attractors, their lengths, and its computational powers? If a relationship is found, does a change in task complexity constrain what the averages of those values are? Such a finding can aid in the construction of RBNs by generative genomes, by favoring those which generate an optimal set of attractors for instance.

The number of attractors and their lengths will be calculated by help of the SAT-solver described in section 3.1.4. It allows for reservoirs with a size of up to 26 to be analyzed within reasonable time limits. The task temporal density will be used to investigate the attractor-accuracy relationship, as $n\_nodes = 26$ is on the verge of solving TD5 with 90% accuracy 4.3. Temporal parity would require too large a reservoir (at least 70). Parameters for the experiment are shown in table 4.6

Table 4.6: Parameters for analysis of reservoir dynamics

Parameter	Configuration
Task	Temporal density 3 and 5
Nodes	26
Connectivity	3
Input connectivity	13
Output connectivity	26
Sample size	500

4.3.2 Results

The distributions of the attractors and corresponding lengths for each set of 500 generated RBNs for TD3 and TD5 are shown in figures 4.5a and 4.5c. The 273 RBNs which gained at least 95% accuracy on TD3 are shown in figure 4.5b. For TD5 the threshold is lowered to 85%, with the 116 matching RBNs this criteria shown in figure 4.5d. This as there only were 24 RBNs with an accuracy of at least 90%, a much too low number to say anything useful.

It is difficult to find theoretical estimates for the mean number of attractors and their lengths for RBNs with connectivity other than $K = 1$ [5] and $K = 2$ [24]. The combined distribution of 1000 RBNs (500 from TD3, 500 from TD5) is therefore used to gain an empirical intuition of what the expected values might be, and is shown in figure 4.5e.

The means of the attractor lengths and corresponding number of attractors for each of the figures in 4.5 are tabulated in table 4.7.

Table 4.7: Means and medians of the different RBN subsets' attractor lengths and number of attractors

	Minimum	**Attractor length**		**Number of attractors**	
	Accuracy	Mean	Median	Mean	Median
TD3	95%	13.90	8.67	5.97	5.00
	ALL	13.45	8.50	5.98	5.00
TD5	85%	11.79	7.90	6.20	5.00
	ALL	12.44	7.67	6.09	5.00
TD3+TD5	ALL	12.95	8.12	6.04	5.00

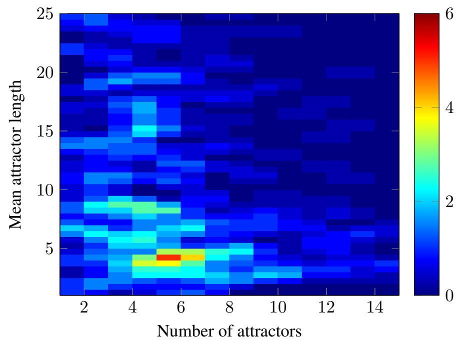
4.3.3 Discussion

First we look at the total distribution of attractors and their lengths in figure 4.5e. Observe from table 4.7 that the median number of attractors and lengths is 5.00 and 8.12 respectively. The most frequent mean attractor length is 3–4 however, with the means being heavily skewed upwards due to a few 256-length outliers. Most RBNs are located in the area of 2–7 attractors and a mean length of 2–8.

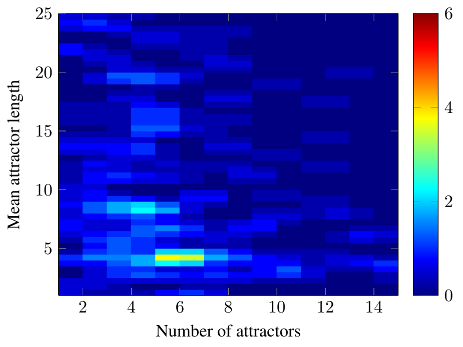
While comparing the figures for TD3 and TD5 against their accuracy-restricted versions next to them, one notices a trend. For both TD3 (figures 4.5a – 4.5b) and TD5 (figures 4.5c – 4.5d) the accuracy restricted versions seem to mimic the overall reservoir distributions for that task. This assumption is confirmed by for each task comparing the means and medians of the restricted versions to those not, in table 4.7. There is but a minuscule difference in the means and medians of TD3 95% versus all TD3, equally so for TD5 85% versus all TD5.

There are two possible conclusions to be drawn from this: For RBNs with $K = 3, p = 0.5, N = 26$, those with 4–5 attractors and mean lengths of 3–4 perform optimally; Or simply that for those parameters, most RBNs would fall within this range. The last conclusion is supported by figure 4.5e. This would mean that a RBNs performance as a reservoir has little correlation to the properties of its attractors.

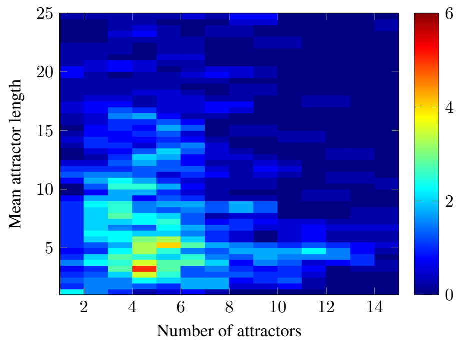
In [27] the authors note that due to continuous perturbation, computation cannot solely depend on the attractors of the system. The system can still be caught by an attractor, but it is unlikely, although not impossible, that it would permanently settle in one. For the RBN RC systems benchmarked in this paper, the large amount of perturbation (50% of the reservoir) would require attractors to have humongous basins to successfully prevent computation. Small attractor basins and large transient times could still be indicators of reservoir performance, as they would allow for more unique paths through state space without immediately leading to an attractor. Chaotic reservoirs are characterized by long transient times and a large sensitivity to perturbations, with states quickly diverging [10]. This could explain why reservoirs with an homogeneous connectivity of three perform



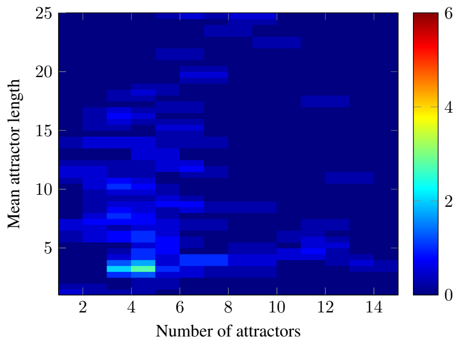
(a) TD3 distribution (500)



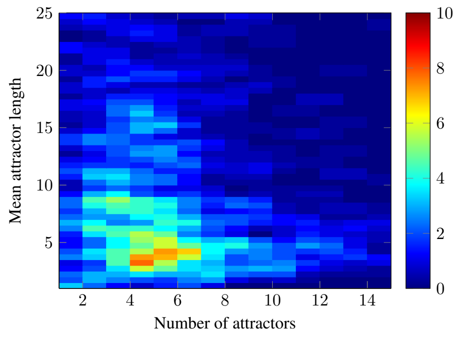
(b) TD3 $accuracy >= 95\%$ (273)



(c) TD5 distribution (500)



(d) TD5 $accuracy >= 85\%$ (116)



(e) Combined distribution of all 1000 RBNs

Figure 4.5: Figures 4.5a and 4.5b show the distribution of the 500 RBNs generated for TD3, and the 273 of 500 that had an $accuracy >= 0.95\%$ on TD3, respectively. Figures 4.5c and 4.5d show the distribution of the 500 RBNs generated for TD5, and the 116 of 500 that had an $accuracy >= 0.85\%$ on TD5, respectively. Figure 4.5e shows the combined distribution of mean attractor lengths and number of attractors for for all 1000 generated RBNs. It is used as an estimate of the common values for RBNs with parameters $K = 3, p = 0.5, N = 26$.

better than those with a connectivity of two. Perturbations would result in large changes to the reservoir state, something that might lessen the burden on the algorithm tasked with finding a suitable mapping to a classification.

CHAPTER

5 CONCLUSION

5.1 Conclusion

Experiments confirm that the required reservoir size increases with the difficulty of the task at hand, with the largest factor being how many bits of input the reservoir is required to remember. Simulation of RBN RC systems can therefore aid in deciding the optimal size of physical reservoirs, given a bridge between the computational power of the reservoir and RBNs can be deduced.

Optimal reservoir perturbation is found to lie at roughly 50% of the size of the reservoir for RBNs with $K = 3$. When using smaller slices of a reservoir for computation, lower amounts of total perturbation will be required as long as these perturbations are located within the same topological area.

Results also show that subsets of larger reservoirs will perform at least as well as a separate reservoir of equal size. Any interference from the unused parts of the reservoir is either minimal or slightly positive.

Finally, no relationship is found between the attractors of a RBN and its performance in a RBN RC system. It can therefore not be used for guiding the construction of accurate RBNs.

5.2 Further Work

When using a computational substrate such as living rat neurons [4], the connections and size of the substrate may change over time. It would be interesting to study how much of a RBN in a RC system can be manipulated before its predictive power collapses, and the readout layer has to be retrained. These changes would include flipping bits in transition tables and changing internal edges in the network. If one can relate the network's robustness to perturbations to a regression model's resistance to variance in the predictor variables, a metric of RBN RC robustness can be developed. It would relate the amount of change in the RBN to the expected reduction of reservoir accuracy.

Another question is in what degree does a reservoirs optimal input connectivity change with regards to what physical subset of the substrate is used? The topology of a RBN is randomly generated and doesn't inherently have a physical mapping, so one would have to be created for this experiment. In the water bucket RC system [9], one would expect that perturbing one part of the reservoir would result in a larger effect in that area rather than the other side of the bucket. If one were to observe this smaller part of the total reservoir only, the hypothesis would be that the perturbation within this restricted area would have to be smaller than if one were to use the entire reservoir for computation.

BIBLIOGRAPHY

[1] Andrew Berdahl et al. “Random sampling versus exact enumeration of attractors in random Boolean networks”. In: *New Journal of Physics* 11.4 (2009), p. 043024.

[2] Nils Bertschinger and Thomas Natschläger. “Real-time computation at the edge of chaos in recurrent neural networks”. In: *Neural computation* 16.7 (2004), pp. 1413–1436.

[3] Aleksander Vognild Burkow. *Evolving Functionally Equivalent Reservoirs for RBN Reservoir Computing Systems*. Pre-master thesis project at NTNU. `https://burkow.no/uploads/forprosjekt-report.pdf`, 2015.

[4] Thomas B DeMarse and Karl P Dockendorf. “Adaptive flight control with living neuronal networks on microelectrode arrays”. In: *Neural Networks, 2005. IJCNN’05. Proceedings. 2005 IEEE International Joint Conference on*. Vol. 3. IEEE. 2005, pp. 1548–1551.

[5] Barbara Drossel, Tamara Mihaljev, and Florian Greil. “Number and length of attractors in a critical Kauffman model with connectivity one”. In: *Physical Review Letters* 94.8 (2005), p. 088701.

[6] Elena Dubrova and Maxim Teslenko. *A SAT-based algorithm for finding attractors in synchronous boolean networks*. `https://people.kth.se/~dubrova/bns.html`. Accessed: 2016-06-14.

[7] Elena Dubrova and Maxim Teslenko. “A SAT-based algorithm for finding attractors in synchronous boolean networks”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)* 8.5 (2011), pp. 1393–1399.

[8] Sigve Sebastian Farstad. “Evolving Cellular Automata in-Materio”. MA thesis. `http://hdl.handle.net/11250/2387463`: NTNU, 2015.

[9] Chrisantha Fernando and Sampsa Sojakka. “Pattern recognition in a bucket”. In: *Advances in artificial life*. Springer, 2003, pp. 588–597.

[10] Carlos Gershenson. “Introduction to random Boolean networks”. In: *arXiv preprint nlin/0408006* (2004).

[11] Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[12] H. Jaeger. “Echo state network”. In: *Scholarpedia* 2.9 (2007). revision #151757, p. 2330.

[13] Herbert Jaeger. “Adaptive nonlinear system identification with echo state networks”. In: *Advances in neural information processing systems*. 2002, pp. 593–600.

[14] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the” echo state network” approach*. GMD-Forschungszentrum Informationstechnik, 2002.

[15] Stuart A Kauffman. “Metabolic stability and epigenesis in randomly constructed genetic nets”. In: *Journal of theoretical biology* 22.3 (1969), pp. 437–467.

[16] Chris G LANGTON. “COMPUTATION AT THE EDGE OF CHAOS: PHASE TRANSITIONS AND EMERGENT COMPUTATION”. In: *Space* 3.5 (), pp. 27–28.

[17] Mantas Lukoševičius, Herbert Jaeger, and Benjamin Schrauwen. “Reservoir computing trends”. In: *KI-Künstliche Intelligenz* 26.4 (2012), pp. 365–371.

[18] *Masters thesis code repository on GitHub*. `https://github.com/aleksanb/MastersThesis`.

[19] Alan Miller. *Subset selection in regression*. CRC Press, 2002.

[20] Julian F Miller and Keith Downing. “Evolution in materio: Looking beyond the silicon box”. In: *Evolvable Hardware, 2002. Proceedings. NASA/DoD Conference on*. IEEE. 2002, pp. 167–176.

[21] Thomas Natschläger, Wolfgang Maass, and Henry Markram. “The” liquid computer”: A novel strategy for real-time computing on time series”. In: *Special issue on Foundations of Information Processing of TELEMATIK* 8.LNMC-ARTICLE-2002-005 (2002), pp. 39–43.

[22] Gordon Pask. “Physical analogues to the growth of a concept”. In: *Mechanization of Thought Processes, Symposium*. Vol. 10. 1958, pp. 765–794.

[23] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[24] Björn Samuelsson and Carl Troein. “Superpolynomial growth in the number of attractors in Kauffman networks”. In: *Physical Review Letters* 90.9 (2003), p. 098701.

[25] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. “An overview of reservoir computing: theory, applications and implementations”. In: *Proceedings of the 15th European Symposium on Artificial Neural Networks*. 2007, pp. 471–482.

[26] Moshe Sipper. “The emergence of cellular computing”. In: *Computer* 32.7 (1999), pp. 18–26.

[27] David Snyder, Alireza Goudarzi, and Christof Teuscher. “Computational capabilities of random automata networks for reservoir computing”. In: *Physical Review E* 87.4 (2013), p. 042808.

[28] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[29] Stephen Wolfram. *A new kind of science*. Vol. 5. Wolfram media Champaign, 2002.

[30] Ozgur Yilmaz. “Reservoir Computing using Cellular Automata”. In: *arXiv preprint arXiv:1410.0162* (2014).

Appendix

A Background chapter bonus content

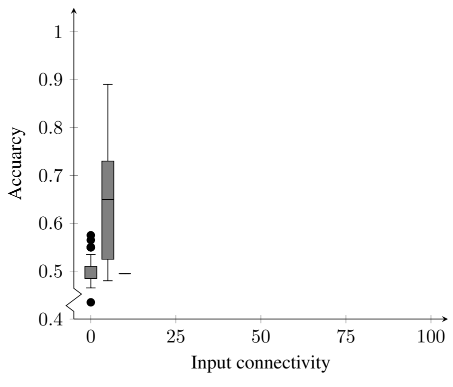
Table 1: Task parameters for the pre-thesis project.

Parameter	**Configuration**
Task type	Temporal parity
Training dataset length	4 000
Test dataset length	200
N (window size)	3 and 5
t (offset)	0

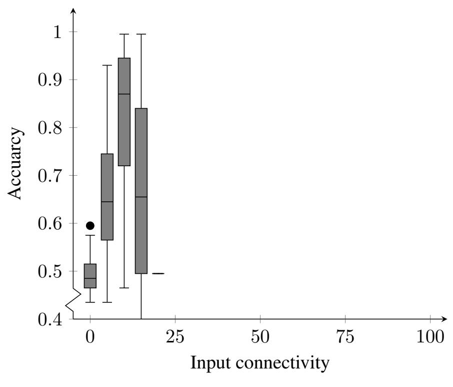
B Minimum required reservoir size, optimal input connectivity

Table 2: Optimal input connectivity values for task / reservoir size combinations as identified from figures 1 through 7. A value of '-' means that the reservoir size wasn't used for that task.

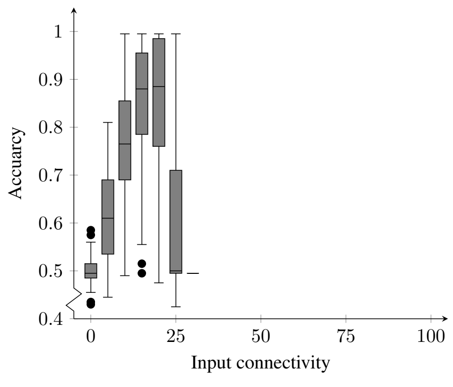
N	TP3	TP5	TD3	TD5
5	-	-	2	-
10	5	-	5	5
15	-	-	5	5
20	10	-	10	10
25	-	-	10	10
30	15	15	15	15
35	-	-	-	15
40	15	20	-	15
45	-	-	-	20
50	30	20	-	25
55	-	-	-	25
60	30	30	-	25
65	-	-	-	30
70	40	30	-	-
80	50	40	-	-
90	50	45	-	-
100	55	50	-	-
110	-	55	-	-
120	-	60	-	-
130	-	70	-	-
140	-	70	-	-



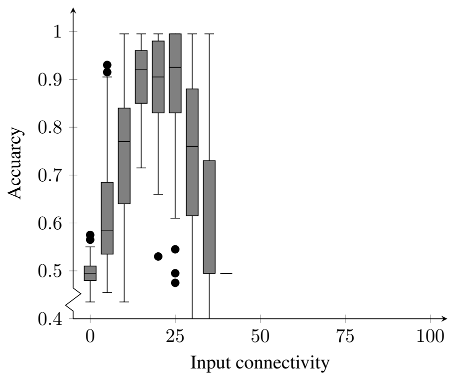
(a) N=10



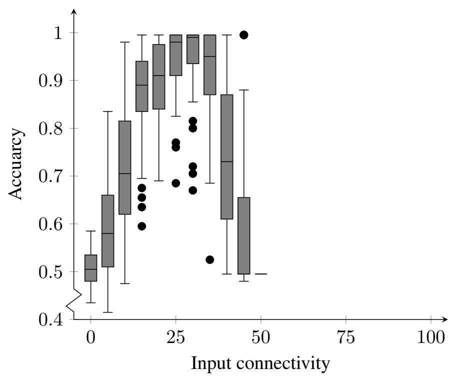
(b) N=20



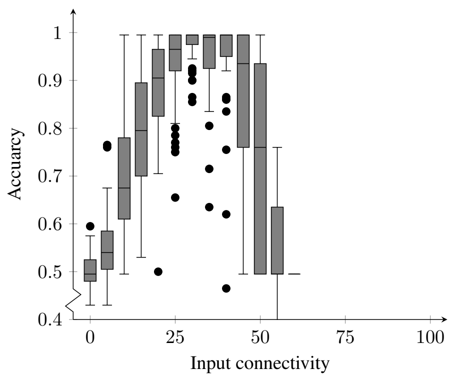
(c) N=30



(d) N=40

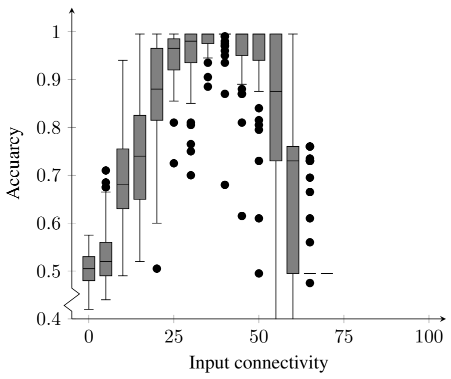


(e) N=50

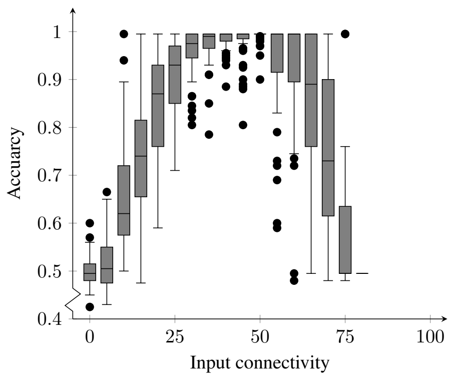


(f) N=60

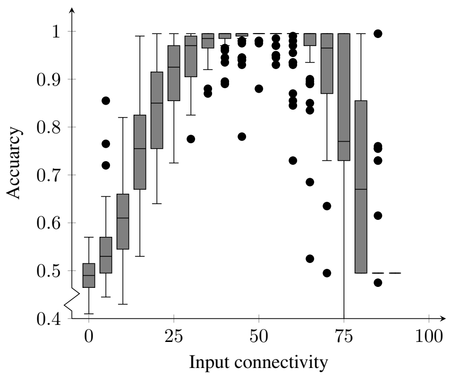
Figure 1: Plots for input connectivity against accuracy for temporal parity 3 (1 of 2)



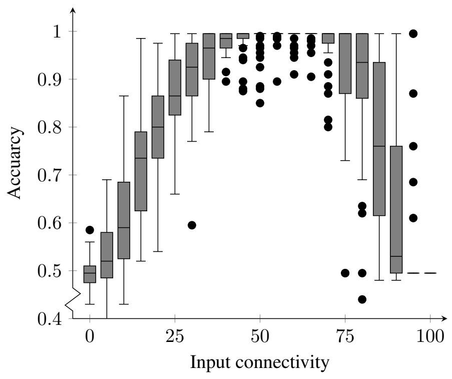
(a) N=70



(b) N=80

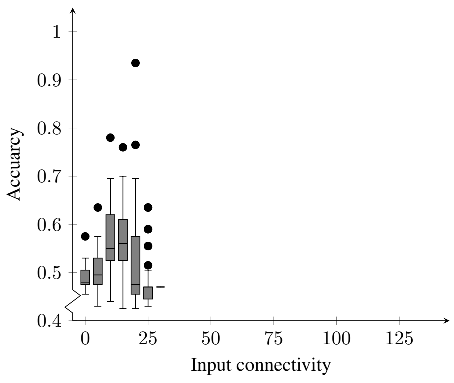


(c) N=90

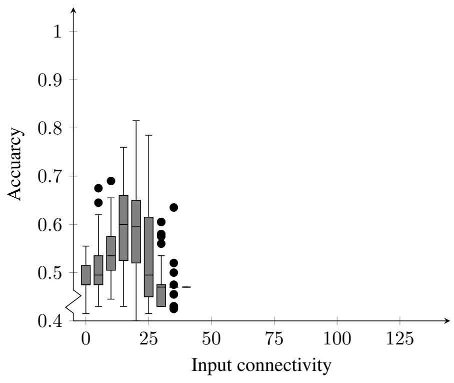


(d) N=100

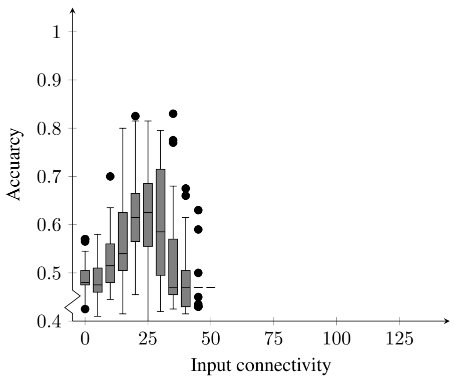
Figure 2: Plots for input connectivity against accuracy for temporal parity 3 (2 of 2)



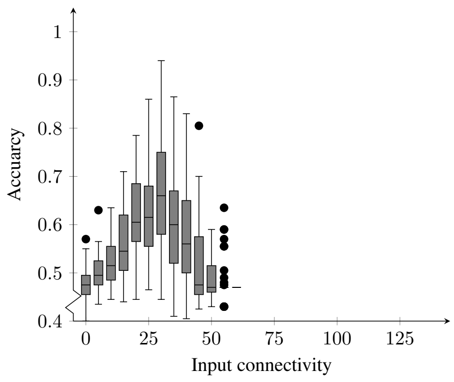
(a) N=30



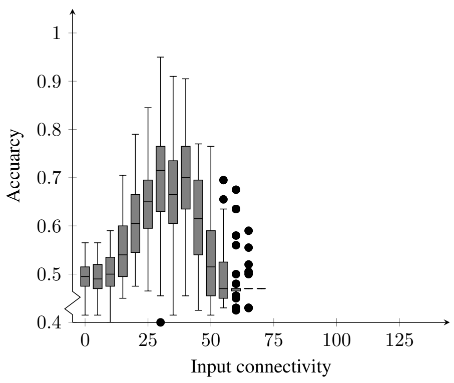
(b) N=40



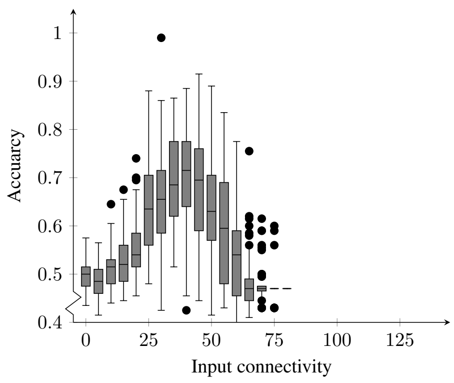
(c) N=50



(d) N=60

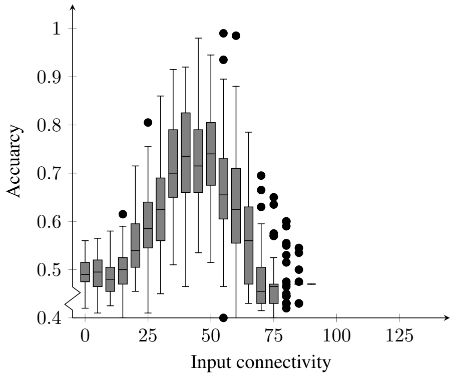


(e) N=70

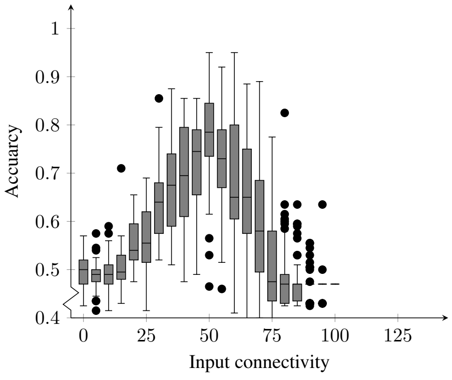


(f) N=80

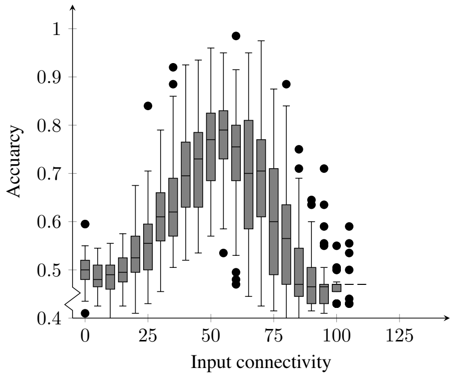
Figure 3: Plots for input connectivity against accuracy for temporal parity 5 (1 of 2)



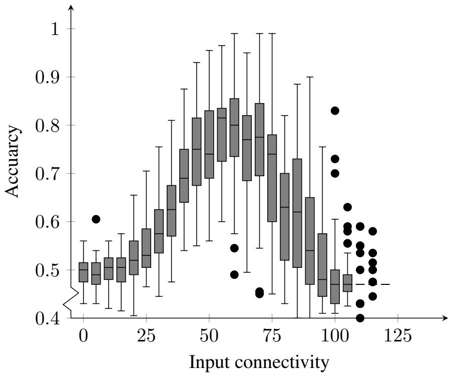
(a) N=90



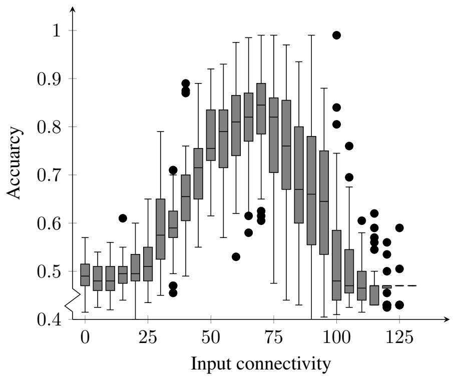
(b) N=100



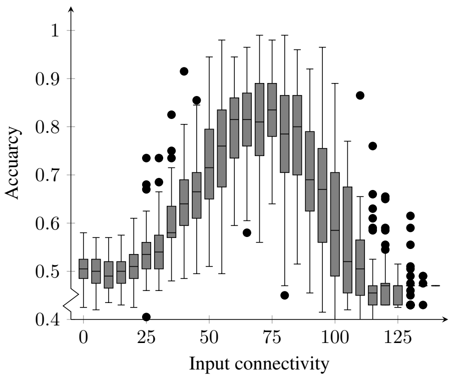
(c) N=110



(d) N=120

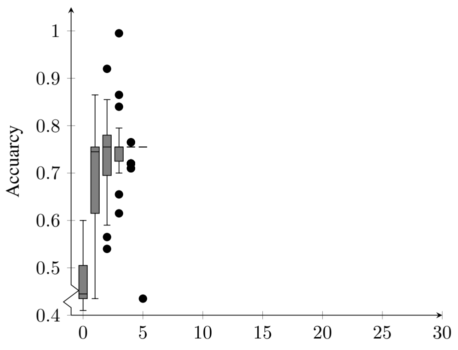


(e) N=130

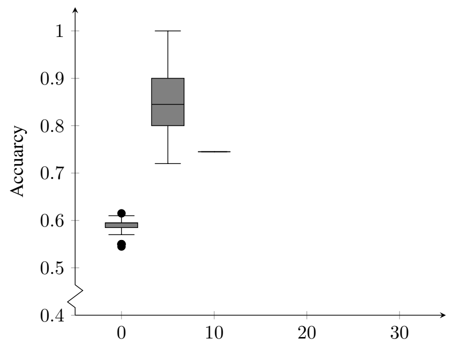


(f) N=140

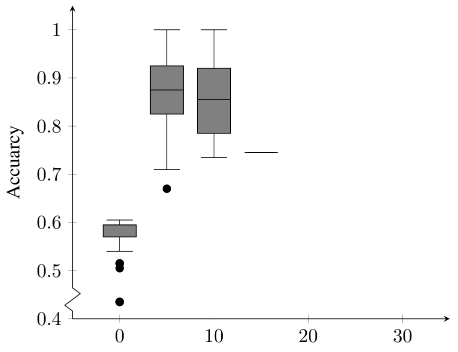
Figure 4: Plots for input connectivity against accuracy for temporal parity 5 (2 of 2)



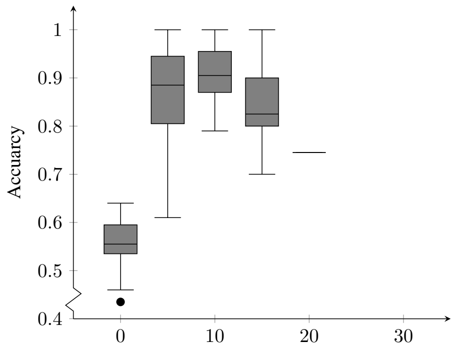
(a) N=5



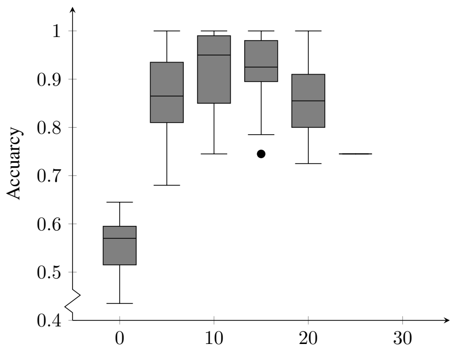
(b) N=10



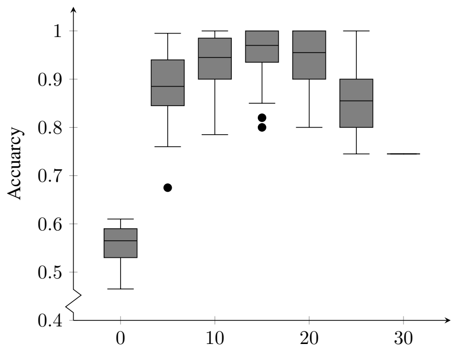
(c) N=15



(d) N=20

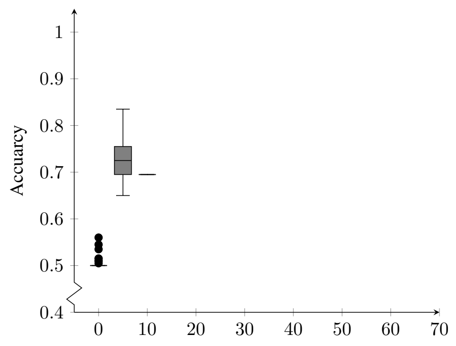


(e) N=25

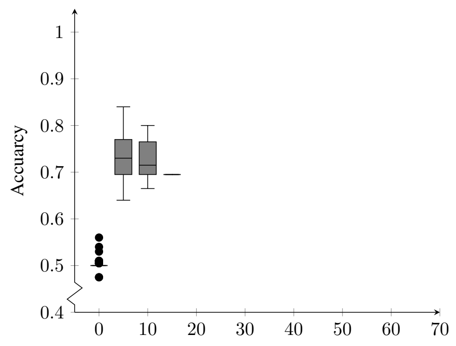


(f) N=30

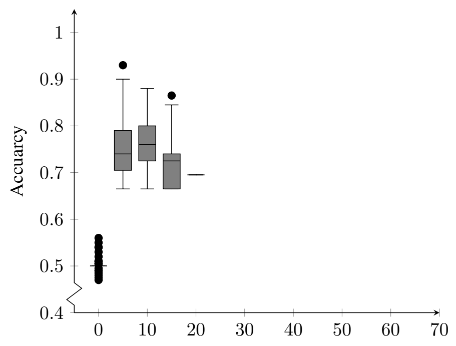
Figure 5: Plots for input connectivity against accuracy for temporal density 3



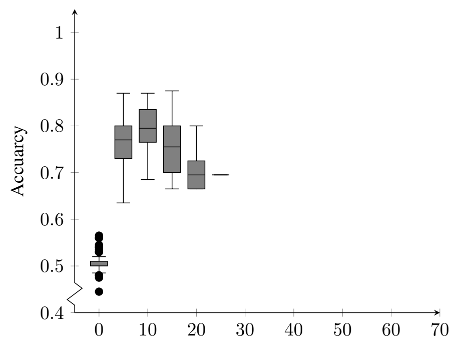
(a) N=10



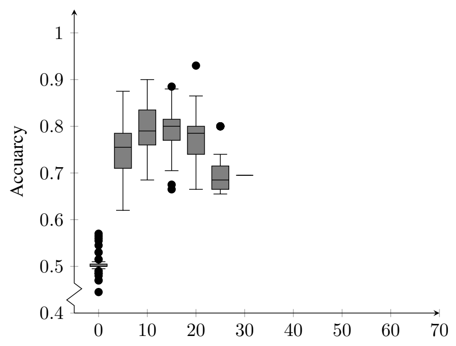
(b) N=15



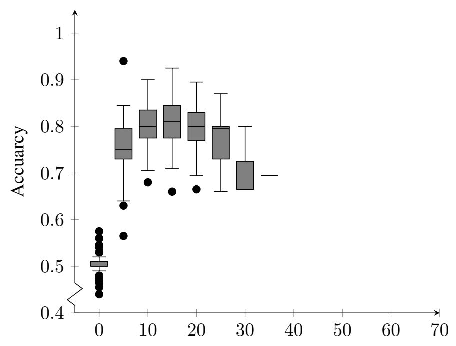
(c) N=20



(d) N=25

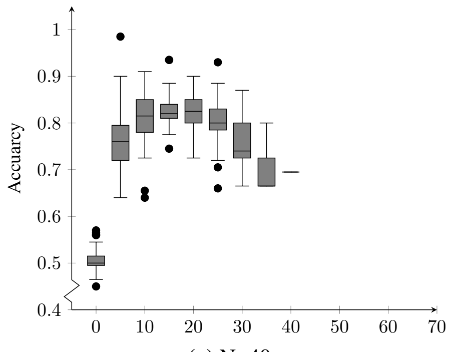


(e) N=30

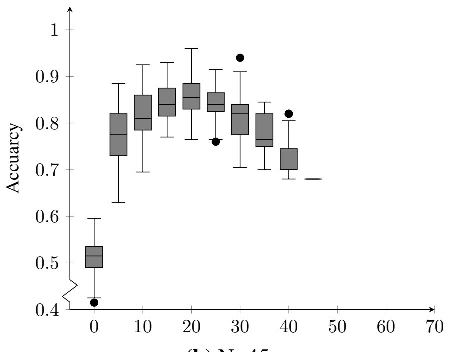


(f) N=35

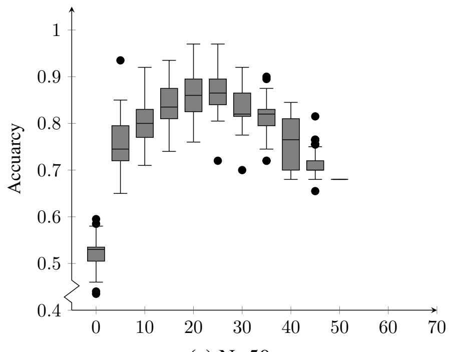
Figure 6: Plots for input connectivity against accuracy for temporal density 5 (1 of 2)



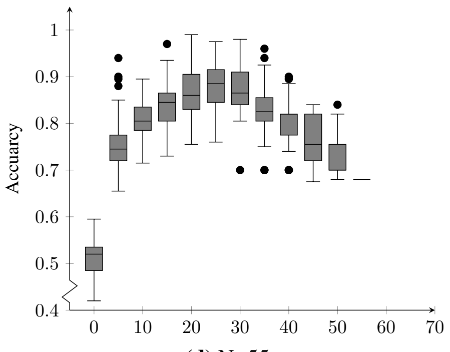
(a) N=40



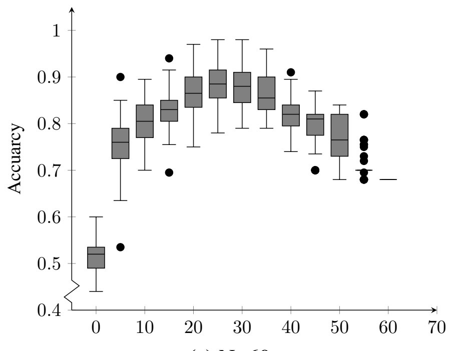
(b) N=45



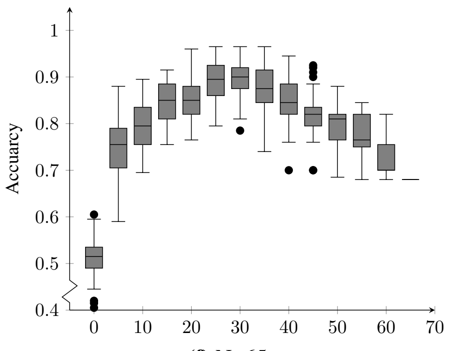
(c) N=50



(d) N=55



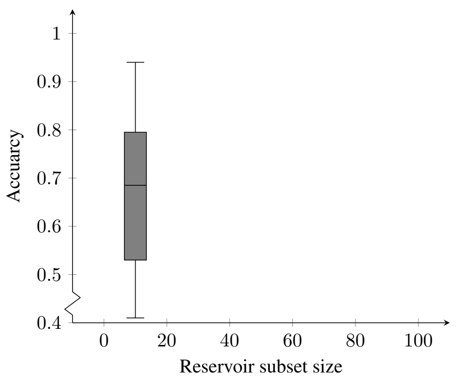
(e) N=60



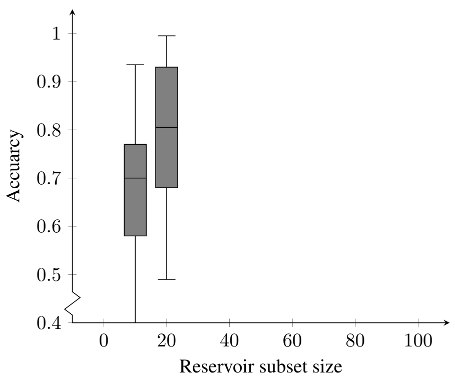
(f) N=65

Figure 7: Plots for input connectivity against accuracy for temporal density 5 (2 of 2)

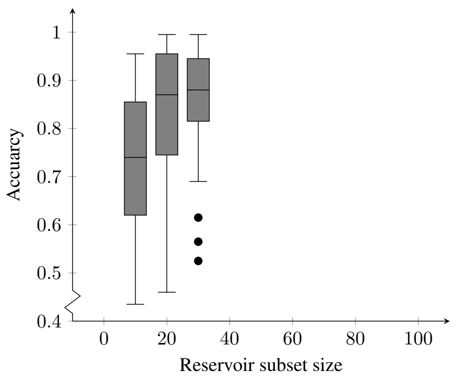
C Minimum required output connectivity



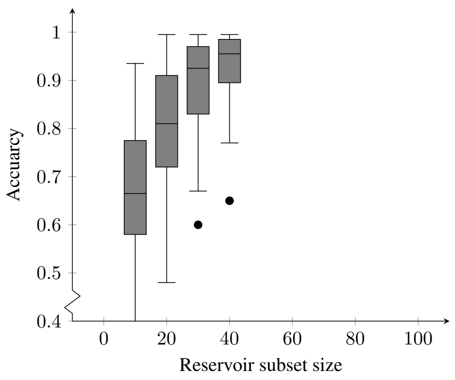
(a) N=10



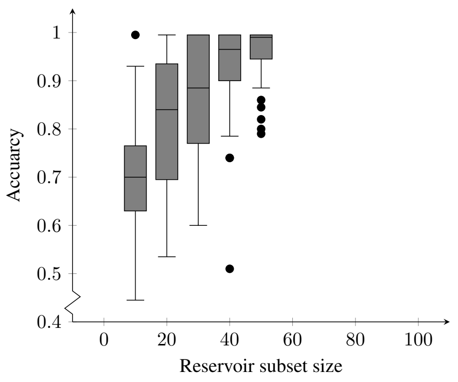
(b) N=20



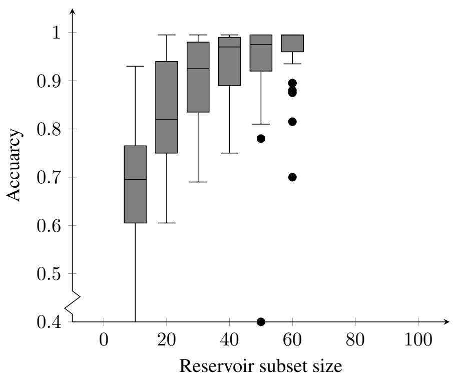
(c) N=30



(d) N=40

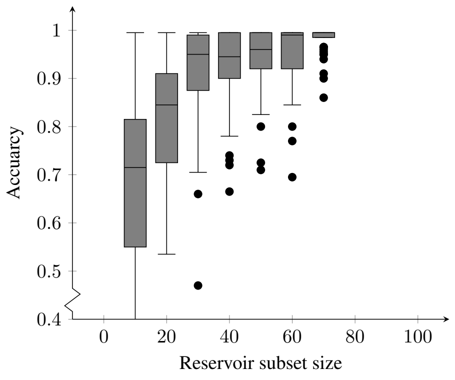


(e) N=50

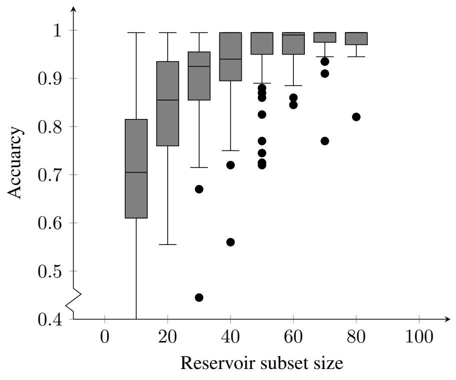


(f) N=60

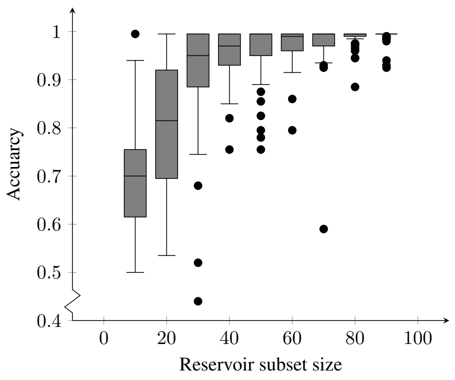
Figure 8: Plots for reservoir subset size against accuracy for temporal parity 3 (1 of 2)



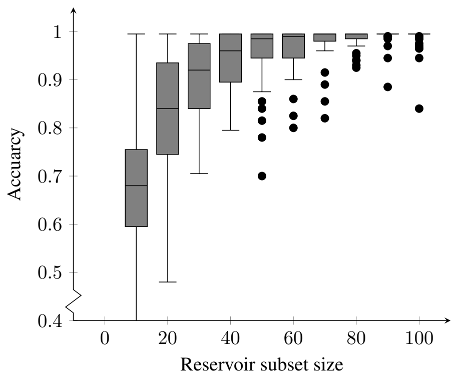
(a) N=70



(b) N=80

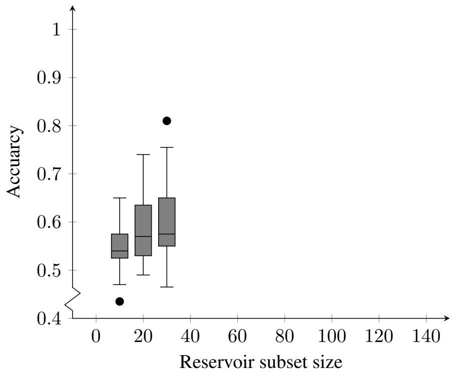


(c) N=90

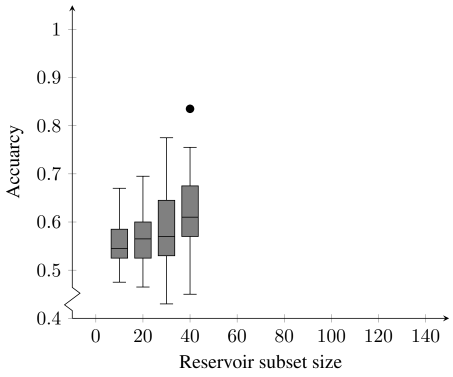


(d) N=100

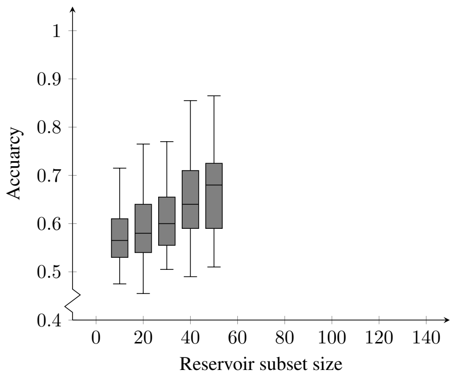
Figure 9: Plots for reservoir subset size against accuracy for temporal parity 3 (2 of 2)



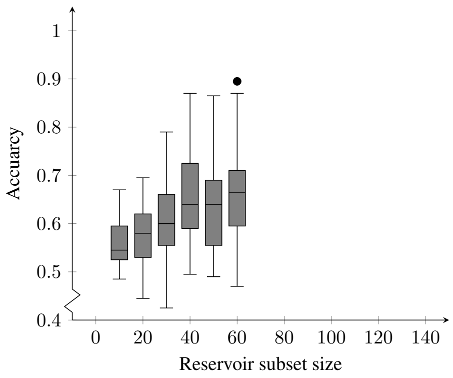
(a) N=30



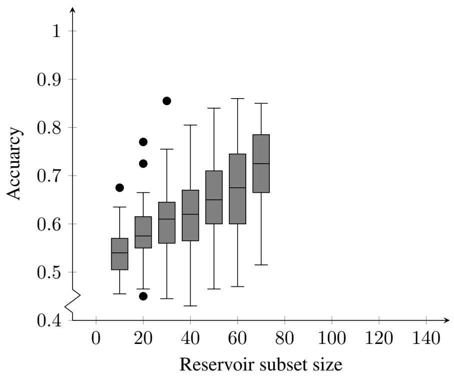
(b) N=40



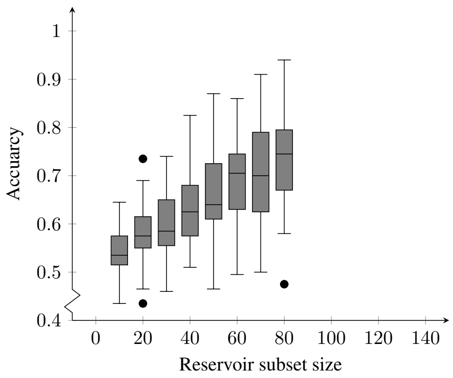
(c) N=50



(d) N=60

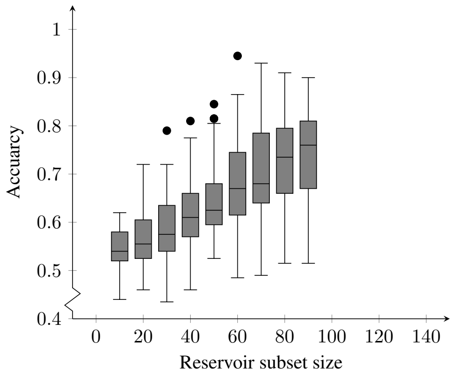


(e) N=70

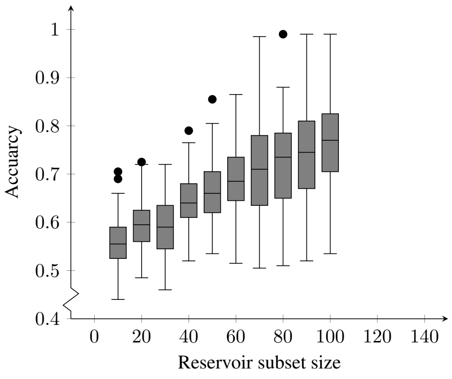


(f) N=80

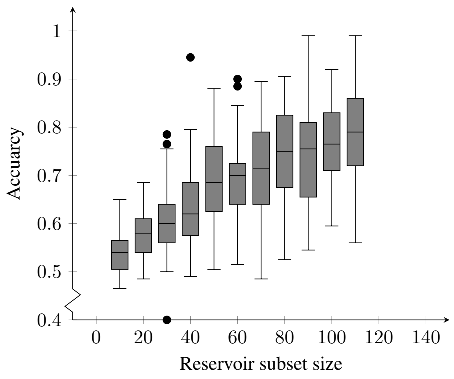
Figure 10: Plots for reservoir subset size against accuracy for temporal parity 5 (1 of 2)



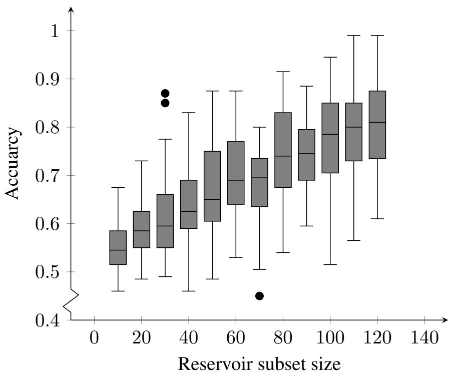
(a) N=90



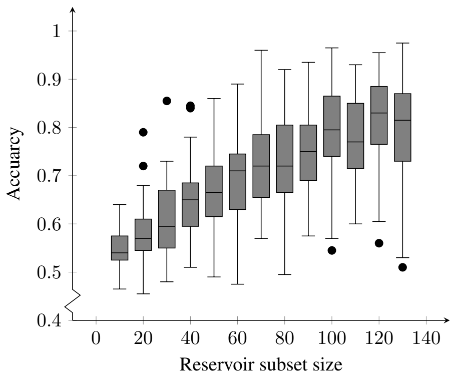
(b) N=100



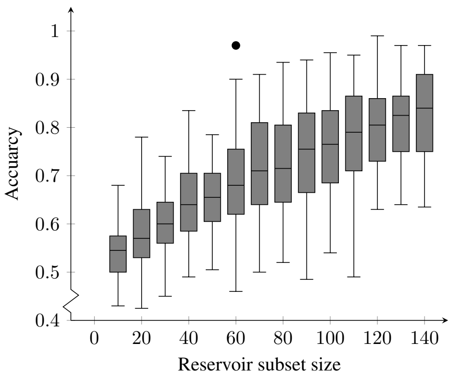
(c) N=110



(d) N=120



(e) N=130



(f) N=140

Figure 11: Plots for reservoir subset size against accuracy for temporal parity 5 (2 of 2)