Evolving Functionally Equivalent Reservoirs for RBN Reservoir Computing Systems

Aleksander Vognild Burkow
Department of Computer and Information Science
Norwegian University of Science and Technology
Sem Sælandsvei 7-9, 7491 Trondheim, Norway
aleksanb@stud.ntnu.no

***Abstract*—Reservoir Computing, a relatively new approach to machine learning, utilizes untrained Recurrent Neural Nets as a reservoir of dynamics to preprocess some temporal task, making it separable with a linear readout layer Originating from the study of Liquid State Machines and Echo State Networks, potentially any sparsely connected network containing feedforward and feedback loops can be a reservoir. Random Boolean Networks (RBN) is such a sparsely connected network that may be suitable for Reservoir Computing.**

In this paper we investigate the dynamics, performance, and viability of RBNs used for Reservoir Computing (RRC). A system to investigate these properties is implemented, and its correctness is validated by comparing its results with those of comparable studies. The chosen reproduced experiments result in the following findings: The more chaotic the phase of an RBN is, the higher its required input connectivity. The value of K which provides optimal computational power is found to lie closer to $K = 3$ when using homogenous networks, as opposed to the heterogenous optimal $\langle K \rangle = 2$. A relationship between Computational Capability and actual reservoir performance seems to exist.

Finally, we find a one-to-many mapping between the readout layer in an already-trained RRC system and different RBN reservoirs, with there being a seemingly large set of interchangeable reservoirs for each readout layer. This makes the potential use of a smaller generative genome for evolving RRC systems interesting. Even though it hits fewer points in the RBN fitness landscape than the fixed genome used in this paper, a large amount of these points are still usable for each instance of a working readout layer.

I. Introduction

Reservoir Computing (RC) is a form of machine learning that sprang out from the study of recurrent neural networks (RNNs). In short, it utilizes the dynamics of some complex system dubbed a 'reservoir' to preprocess a timeseries problem, transforming it from a temporal to a spacial one in the reservoir, making it then separable with a usually simple readout layer.

In this paper the dynamics of Reservoir Computing systems where the reservoir is a Random Boolean Network (RBN) [5] is investigated. In [14] RBN Reservoir Computing systems (RRC) were investigated, and found to be a fruitful approach. A computational system consisting of such simple nodes with inherent emergent properties are an interesting field of study as an alternative to classical computation. The study of such networks can also pave the way for selecting physical substrates as reservoirs.

First we create a working RBN Reservoir Computing system in the Python programming language, reproducing chosen experiments from [14]. These include finding the optimal input connectivity (L) and internal connectivity K, as well as testing their measure of Computational Capability against actual reservoir performance.

Next we investigate whether the readout layer of a working RRC system can be re-used with other RBN-reservoirs than the one it was originally trained on, and still stay accurate on the original classification task. These functionally equivalent reservoirs, if any, will be evolved through the use of a genetic algorithm (GA).

Finally we look at the dynamics and characteristics of these groups of RBN Reservoirs, attempting to find any similarities that made them exploitable for computation.

II. Background

A. A Brief Introduction to Reservoir Computing

Recurrent Neural Networks, as opposed to feed-forward neural networks, are notoriously time consuming and difficult to train. 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.

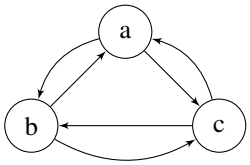
It was therefore proposed both in [7] (as Echo State Networks, or ESN) and [12] (as Liquid State Machines, or LSM) to separate the RNN into two parts, the untrained reccurrent reservoir, and the trained readout layer. Both of these methods have been unified into the field of Reservoir Computing, now focusing on the separate training and evolution of the recurrent and readout parts [10].

Exiting applications of Reservoir Computing include speech and handwriting recognition, as well as controlling robotics [10].

B. Alternatives to classical reservoirs

Are there other types of complex systems that can be used as reservoirs? What properties must these reservoirs have to be able to solve problems?

Complex networks similar to the sparsely connected RNNs used ESN and LSM systems include Cellular Automata and Random Boolean Networks.



(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 1. An example homogenous RBN with $N = 3, K = 2, P = 0.5$.

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 will be introduced in depth in II-C.

Both models are simple, and can be implemented in software, hardware (FPGAs), and in materio [11] (for evolving CAs in materio, see [2]). 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 [13].

The 'Water bucket' paper investigated the use of an actual bucket of water as a reservoir [3], successfully recognizing patterns and achieving decent performance at that. The RBN Reservoir approach has also been found to be viable [14] .

C. A Brief Introduction to Boolean Networks

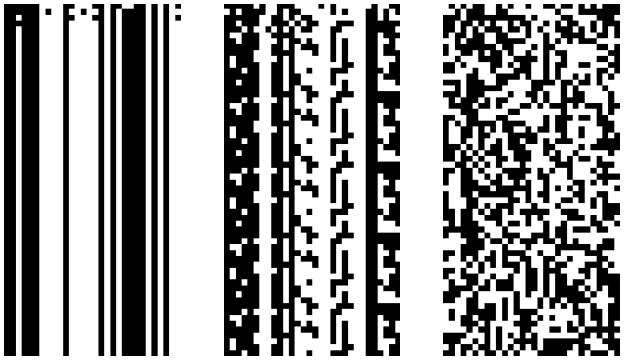
Random Boolean Networks, also known as Kaffman networks, were originally developed as a model of gene regulatory networks [8], the complex system that regulates how genes in multicellular organisms interact with each other. The model requires no assumptions about the inner workings of the actual nodes, 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 multi-valued 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 homogenous and heterogenous in-degrees. In heterogenous 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 1 visualizes a homogenous 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 timestep $t + 1$ therefore only depend on the states at the previous timestep t. A criticism of the classical model is that gene regulation networks are updating continiously, as opposed to in lockstep.c



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

Figure 2. 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.

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.

The dynamics of an 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 [5]. 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.

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 [9]. Critical systems are found on the edge of chaos, on the phase transition between ordered and chaotic networks [5]. For RBNs with $\langle p \rangle = 0.5$, critical dynamics are usually found at $\langle K \rangle = 2$ [5], 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 [5].

D. RBN Reservoir 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 continiously 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

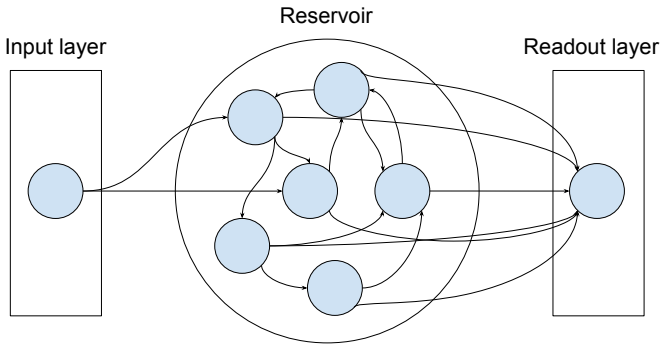
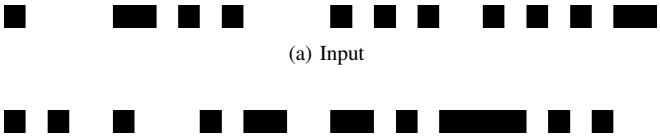


Figure 3. RBN-Reservoir system with $I = 1, L = 2, K = 2, N = 5$. The reservoir transforms the problem from a temporal one to a multidimentional spatial one. The readout layer the performs some kind of learning on the reservoir states against the expected output for the current task.



(a) Input

(b) Correct output

Figure 4. 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 $[i, i-1, i-2]$

perturbed by the input at each timestep, and what dynamics must such a reservoir have to allow for the computation of interesting problems?

1) A working system: In [14] the authors create and analyze functioning RBN-RC systems. These RBN-RC systems have heterogenous connectivity, consume one bit of input at each timestep ($I = 1$), perturbing L 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 outout for the current task, e.g. linear regression. Such a setup is shown in Figure 3.

2) Tasks: To measure the real-life performance and accuracy of the RBN Reservoir systems, two tasks were introduced: Temporal Density and Temporal Parity [14]. Both require the reservoir to be able to retain information for a sliding window of size n, offset by some value t, back 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 4, and will be used to benchmark the reservoirs created later in this paper.

3) Computational capability: For an 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 [1].

These two properties are coined *fading memory* and *separation property*, and can be measured [14] 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 an RBN-reservoir is then defined as

$$\Delta_{Tt} = separation\_property_{Tt} - fading\_memory_{T} \quad (1)$$

Analyzing different RBN-reservoirs with this metric [14], 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 ($t = 0$).

4) Optimal perturbance: It is found that the optimal amount of reservoir perturbance, 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.

E. Genetic Algorithms

A genetic algorithm is an advanced probabilistic search method using operations inspired by natural evolution to traverse the solution space of the problem. It mimics natural evolution by creating a population of children where some will grow up to adults, the fittest of which reproduce the most. The newly created offspring receives a genome derived from its parents, using a combination of genome crossover and random mutations to push the search forward.

Good choices for genome representations and fitness functions make GAs excellent for finding solutions to optimisation problems. Good hyperparameters (parameters for the GA run itself, i.e. genome crossover rate, adult and child population sizes) may differ widely across problem domains and genome representations. If an certain GA instantiation performs well in some domain, it must necessarily perform worse on others as proven by the No Free Lunch Theorem [16].

III. Method

To verify the viability of RBNs in RC systems, a functioning RRC system has to be created. Being able to reproduce results from [14] will lend credibility to the approach presented in this paper. The Computational Capability measure presented therein will be used to analyze our RRC systems.

Second, we wish to investigate the potential many-to-one relationship betwen reservoirs and previously trained readout layers, evolving these functionally equivalent reservoirs

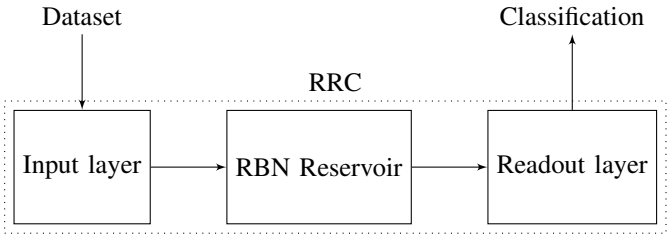


Figure 5. Block diagram of the RRC processing a dataset.

through artificial evolution. These sets of interchangeable reservoirs may share similar attributes and Computational Capability, and will be analyzed. If such a mapping exists it may tell us something about how large a part of the RBN fitness landscape is usable as a reservoir, and how difficult it is to reach these points. A smaller generative genome could then be used instead of the fixed one presented in this paper, potentially guided towards the attributes we find useful.

The following systems have been implemented to investigate the stated questions:

- An RBN simulator
- Procedures for analyzing and visualizing RBNs
- Procedures for creating classification tasks
- An RBN Reservoir Computing system using:
  - The aforementioned RBN simulator as a reservoir
  - The ridge regression node from the Oger RC toolkit [15] as readout layer
  - The Python Modular Toolkit for Data Processing [17] for glue and training of the RRC system.
- A system for evolving RBNs given certain constraints, using a genetic algorithm based on the one used in [2].

The codebase is available on GitHub [4] under a soon-to-be permissive licence.

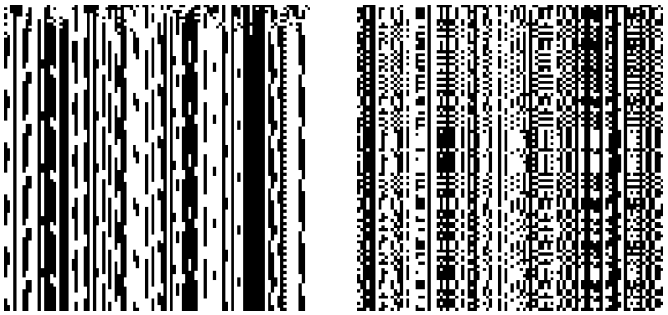
A. Measuring Computational Capability

We will be using the computational complexity measure introduced in II-D3 to analyze the created RBNs. This measure is parameterized over both input stream length T and the required depth of separation t. To make this an useful measure when comparing against reservoir accuracy on a specific task, we will chose values of T similar to the length and t equal to the required memory of that task.

B. The creation and training of a functioning RRC system

The final RRC system is shown as a block diagram in Figure 5, and the actual network topology is equivalent to the one in Figure 3.

1) 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 6a. We see that the RBN exhibits stable dynamics, and enters into an attractor around $t = 15$. In Figure 6b we continiously perturb the RBN with the input stream from the Temporal Parity task visualized in Figure 4. In the perturbed case, the state trajectory is continiously 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 dissapear with a RBN in the chaotic phase.



(a) Unperturbed (b) Perturbed

Figure 6. The same RBN ($N = 100, K = 2, P = 0.5, L = 50$) shown both perturbed and unperturbed. The boolean states of the RBN are plotted along the X-axis, with time flowing downwards.

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.

2) Training: To train the RRC system we require a number of training datasets, as well as different testing datasets to test the trained system. We will use the datasets described in section II-D.

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 perturbance, the RBN is ran synchronously (CRBN mode) for one timestep. 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 [6] is used. This version of least squares regression is more accurate when faced with input colinearities, 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

$$Accuracy = 1 - \frac{sum(actual\_output \neq expected\_output)}{len(correct\_output)} \tag{2}$$

. If the RRC system achieves a high accuracy on an interesting task, it will be stored for further research.

C. The evolving of functionally equivalent RBN reservoirs for existing readout layers

To investigate the potential many-to-one mapping between RBN-reservoirs and readout layers, a Genetic Algorithm (sec-

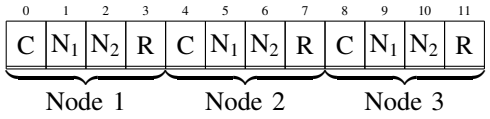


Figure 7. The direct-encoded genotype used for evolving RBNs, shown here for an RBN with $N = 3, K = 2$.

Table I
GA HYPERPARAMETERS

Children pool size	40
Adult pool size	40
Fitness satisfaction threshold	0.98
maximum generations	200
Adult selection	Generational mixing
Parent selection	Tournament selection(K=8)
Genome crossover	Per component crossover (p=0.5)
Genome mutation	Per genome component mutation (p=0.1)

tion II-E) will be used. A fixed test dataset and readout layer for fitness evaluation will be used.

1) Genotype and phenotype representation: We let the genotype be a direct encoding of the corresponding RBN graph. Such an encoding is significantly larger than a generative encoding, but less complex as well as able to generate all individuals in the fitness landscape.

Each node needs to represent whether it is connected to the input node, who its neighbors are, and what its transition rule is. As we only look at homogenous RBNs, we can use a fixed-length genome with

$$genome\_length = n\_nodes \cdot (connectivity + 2) \quad (3)$$

To further simplify GA implementation, we let all symbols in the genome take a value in the range $[0, max\_required\_in\_genome)$. The actual symbol values are then computed modulo the largest actual value they could take on (2 for input connectivity, n_nodes for neighbors, $2^{2^{n\_nodes}}$ for transition rules).

The final genome is shown in figure 7. Note that this representation sets no limitations on input connectivity or uniqueness of neighbors.

2) Fitness function: Each GA run is parameterized with an already-trained and accurate readout layer (separated from its RBN-reservoir), as well as a single test dataset of the same origins as the dataset used for training the readout layer. After converting each genotype to its corresponding phenotype, the resulting RBN is connected to the readout-layer and fed with the fixed test dataset. The accuracy obtained, as calculated in formula 2, is used as the fitness of the phenotype.

3) GA hyperparameters: The hyperparameters for the GA are presented in table I.

Adult selection is simple generational mixing, selecting the best 40 specimens from the combined children and adult populations. Per component crossover will with a probability p either pass through the entire genome from the left parent, or for each component chose either the left or right parents value with a probability 0.5. Per genome component mutation will reroll each genome component with probability p. This value is set relatively high (0.1) due to experiments showing a much faster convergence rate fr this domain than with the initial chosen value of 0.01.

Table II
TASK PARAMETERS

Task type	Temporal Parity
Num. datasets	10
Dataset length	200
N (window size)	3 and 5
t (offset)	0

Table III
RBN COMBINATIONS

N (nodes)	100
K (connectivity)	1, 2, 3
L (input connectivity)	0, 10, ..., 100
Temporal parity	$N = 3, 5$

IV. EXPERIMENTS

A. Creating functioning RBN reservoir systems

In [14] the authors find a relationship between computational capability and performance, that certain values of L give the best reservoir performance, and that RBNs with connectivity $\langle K \rangle = 2$ should outperform other connectivities. To test these assertions, we must create and benchmark a number of functioning RBN reservoir systems, noting their accuracy on the chosen task as well as their computational capability. Plotting these values against each other will shed light on whether their claims strike true.

We will be using two versions of the *temporal parity* task (as specified in table II) to measure reservoir performance. The temporal parity task is chosen over temporal density as it is the more difficult task (shown in [14]), presumably resulting in more interesting and rich resevoirs.

Computational Capability will be measured with $T = 100$ for both versions of the task, but the required memory t equal to the tasks window size N.

As the number of different RBNs is oppressively large, $(\frac{2^{2^K} N!}{(N-K)!})^N$ [5], we therefore create 30 random specimens for each combination of the RBN parameters displayed in table III. This results in 300 samples for each RBN N-K combination for each task. Note that the created RBNs are of homogenous connectivity, as opposed to the heterogenous connectivity used [14].

B. Evolving functionally equivalent RBN reservoirs for existing readout layers

We use the genetic algorithm presented in section to evolve functionally equivalent RBNs (hyperparameters as in table I).

We now pick 5 RBN reservoir systems, chosen for their $> 99.5\%$ accuracy on the Temporal Parity task shown in table IV, and store their trained readout layers. All 5 RBNs have $N = 100, K = 2$, three of the RBNs having an input connectivity of $L = 50$, the remaining two $L = 70$. They will from now on

Table IV
GA TASK PARAMETERS

Task type	Temporal Parity
Num. datasets	10
Dataset length	200
N (window size)	3
t (offset)	0

be referred to as L50#1, L50#2, L50#3, L70#1, and L70#2. This will allow us to measure if the functionally equivalent reservoirs for the readout layers trained on reservoirs with $L = 50$ differ from the ones with $L = 70$.

For each readout layer we create a new instance of the Temporal Parity task used to train it originally. We then run the GA 30 times with this readout layer–dataset combination, sampling only the best specimen from each final generation. This to keep our influence from evolutionary inbreeding to a minimum.

V. RESULTS

A. Creating functioning RBN reservoir systems

Figures 8 and 9 contains plots of input connectivity against accuracy, as well as Computational Capability against accuracy for the sampled populations (for Temporal Parity with $N = 3$ and $N = 5$ respectively).

Each plot contains a total of 300 samples, 30 for each input connectivity in the input connectivity boxplots.

1) Temporal Parity $N = 3$: Almost no reservoirs with $K = 1$ (fig. 8a) are able to solve the task adequately, almost all samples having a worse accuracy than 0.5. In the corresponding Computational Capability plot (fig. 8d), all samples have a low CC mapping to a correspondingly low accuracy. There are many reservoirs able to complete the task for $K = 2$ (fig. 8b). The mean population fitness peaks at input connectivities 40–50, giving a firm suggestion to where the optimal connectivity is located. In the corresponding CC plot (fig. 8e), an increase in CC tends to correlate with an increase in mean accuracy.

The mean fitness is even higher for the reservoirs with $K = 3$ (fig. 8b), peaking at the input connectivities of 50–60. The corresponding CC-Accuracy plot in fig. 8f has its distribution skewed upwards towards the right.

2) Temporal Parity $N = 5$: Temporal Parity with $N = 5$ is a considerably more difficult task for RBNs of size 100, as shown in figure 9. Again reservoirs with $K = 1$ (fig. 9a) are unable to complete the task, the abysmal CC-Accuracy plot of fig. 9d telling the same story. Reservoirs with $K = 2$ tell the same story (fig. 9b). There is but a single well-performing outlier at the input connectivity of 50.

The mean fitness is considerably higher for $K = 3$ (fig. 9c), still having its best reservoirs at the input connectivity of 50-60. The CC plot (fig. 9f) contains considerable more samples with higher CC.

B. Evolving functionally equivalent RBN reservoirs for existing readout layers

Out of the total $30 \cdot 5 = 150$ evolved RBNs, all except five achieved an accuracy of at least 99%. In Figure 10 we see plots of the 30 termination generations for each of the five readout layers. The worst accuracy of 78% was achieved in the GA run of L50#3 which timed out at generation 200.

The fitness plots for one of the GA runs of L50#3 is shown in Figure 11. This kind of fitness distribution, where the third quantile peaks at roughly 70% accuracy with the best individual being found not gradually, but with a sudden jump, is representative for most of the GA runs performed.

The input connectivity of the evolved reservoirs is shown in figure 12. Observe that the median of the distributions seem largely independent of the input connectivity of the original reservoir, the majority of its functionally equivalent RBNs centered around $L = 50$.

Last we look at the computational capabilities of the evolved RBNs against the CC of the original reservoirs in figure 13. There does not seem to be a strong relationship betwen the original CC values and the evolved ones. The distribution of L70#2 is shifted towards 0.0, but the distribution of L70#1 is not, even though its value is close to zero as well.

If one looks at the CC-Accuracy plots of Figure 8e, these distributions don't look that out of place. They might simply be a representation of where most reservoirs with $N = 100, K = 2, L = \langle 50 \rangle$ lie.

VI. DISCUSSION

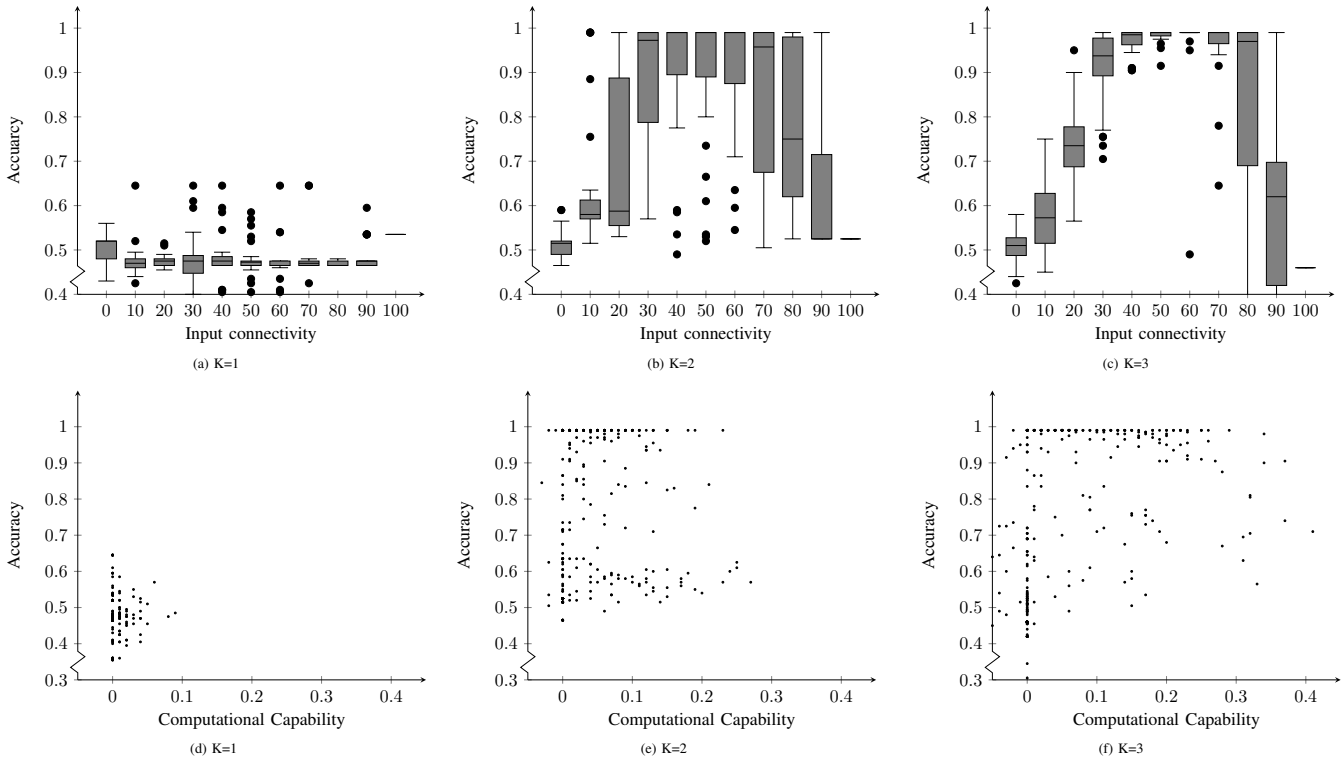
A. Creating functioning RBN reservoir systems

1) Temporal Parity with $N = 3$: There is an abundance of well-performing reservoirs for $K = 2$ and $K = 3$. Reservoirs with higher connectivity have their fitnesses peak at higher values of input connectivity. This is in line with the expectations from [14] described in section II-D4. The more chaotic the dynamics of the reservoir, the more it has to be perturbed to retain the input information.

2) Temporal Parity with $N = 5$: The distribution of well-performing reservoirs is shifted heavily towards $K = 3$. There are single outliers that perform well for $K = 2$, but the accuracy medians are considerably lower than for the reservoirs with $K = 3$.

Comparing the observed performance against the performace of the RRC systems benchmarked in [14] on the same task, one sees the same drop in performance from Temporal Parity with $N = 3$ to $N = 5$.

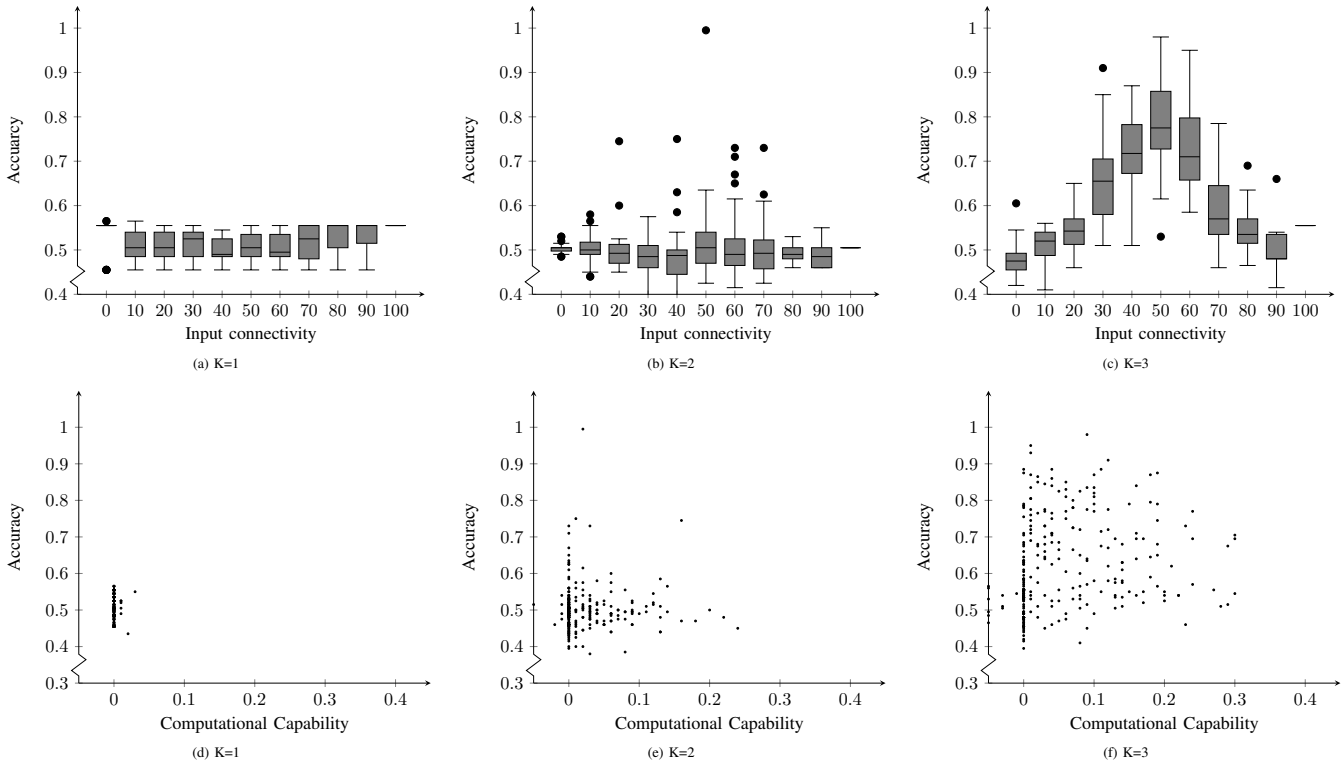
3) Computational Capability: Comparing the Computational Capability plots for both tasks (Figures 8 and 9), one observes that the individuals with a higher Computational Capability have a tendency to be shifted towards higher accuracies. The lower accuracy bounds are increased as compared to the samples located at $.0CC$. This indicates a positive correlation between Computational Capability and performance, again in line with the expectations from [14]. One might be quick to assume that reservoirs achieving no higher than a



(a) K=1 (b) K=2 (c) K=3

(d) K=1 (e) K=2 (f) K=3

Figure 8. Plots for Temporal Parity with $N = 3$. Figures 8a–8c plot the accuracies of the sampled RBNs against their input connectivity, for K=1–3 respectively. Figures 8d–8f plot the accuracy of the previous figures against their Computational Capability ($T = 100, t = 3$).



(a) K=1 (b) K=2 (c) K=3

(d) K=1 (e) K=2 (f) K=3

Figure 9. Plots for Temporal Parity with $N = 5$. Figures 9a–9c plot the accuracies of the sampled RBNs against their input connectivity, for K=1–3 respectively. Figures 9d–9f plot the accuracy of the previous figures against their Computational Capability ($T = 100, t = 5$).

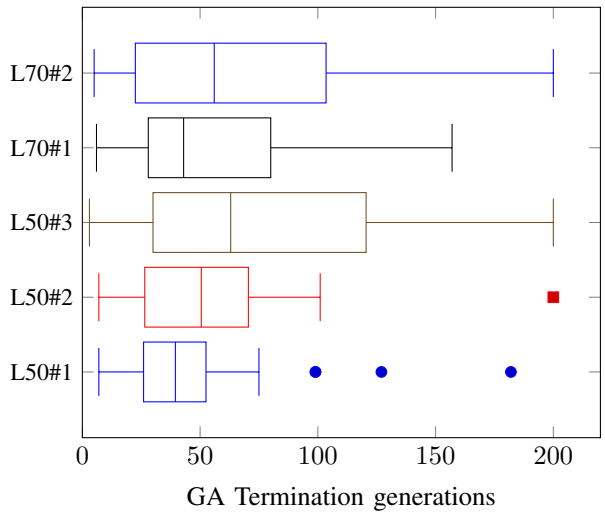


Figure 10. Termination generations for the evolution of functionally equivalent RBNs.

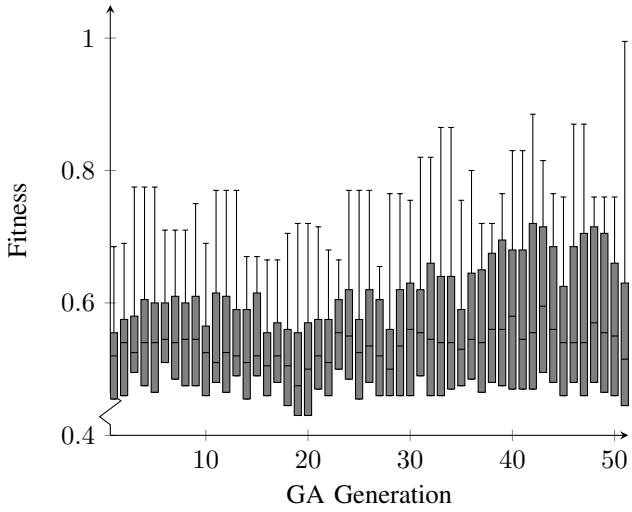


Figure 11. Fitness of the GA population at each generation. The mean stays relatively close to 50% accuracy the whole time, althoug the third quartile pushes upwards towards the end. Due to only storing max fitness and fitness quartiles during simulation, the whiskers represent the best fitness as opposed to the 98th percentile.

0.5 accuracy perform no better than the tossing of a coin. There is however no guarantee that the distribution of correct classifications follow a binomial model with $p = 0.5$.

4) Optimal Connectivity: When comparing general performance across K values on the two tasks, the optimal connectivitity seems to be closer to $K = 3$ than $K = 2$. As described in section II-C, critical dynamics for heterogenous networks should be the most frequent at an average $\langle K \rangle = 2$. Such networks can therefore contain subgraphs of both higher and lower connectivity, while keeping the average connectivity the same. This disparity can therefore be explained by the fact that an homogenous RBN with $K = 3$ can emulate an RBN with lower connectivity by having two or more in-edges from the same ancestor node, while an homogenous RBN with $K = 2$ cannot emulate a higher-connectivity one.

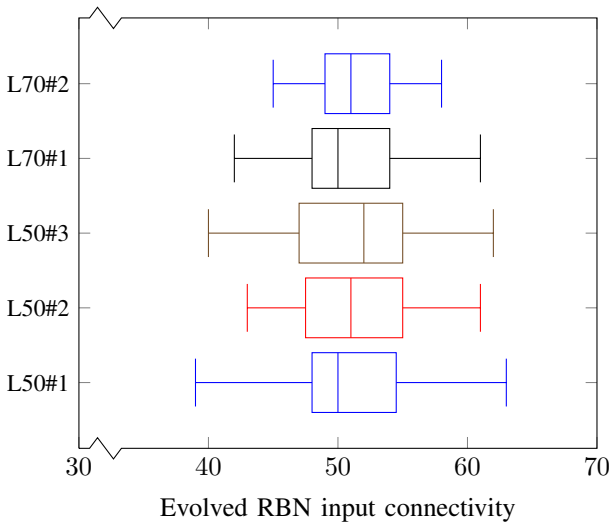


Figure 12. Connectivity for evolved RBNS. Observe that the connectivity distribution of the evolved RBNs is centered heaviliy around 50 regardless of the original input connectivity.

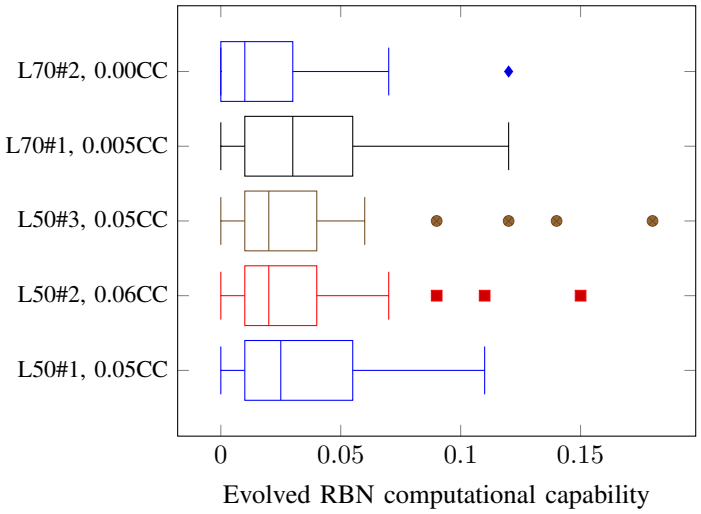


Figure 13. Computational capabilities for the evolved RBNs.

B. Evolving functionally equivalent RBN reservoirs for existing readout layers

There are a great number of functionally equivalent reservoirs for each functioning readout layer, and the Genetic Algorithm finds them efficiently. Sadly, a tight correlation between the properties of the RBN from the original RRC system, and RBNs evolved against its readout layer seem spurious at best. In fact, the connectivity distributions of Figure 12 and computational capabilities of Figure 13 seem much more representative of the general RBN population, as shown in Figure 8b. This indicates that while there are many compatible reservoirs for a given readout layer, the distribution of the reservoirs are likely the same as the distribution of reservoirs with the same connectivity in general.

It should be noted that during earlier simulations with a per-component mutation rate of 1% as opposed to the 10% actually used, the mean of the fitness distribution during GA runs would be much closer to the best specimens, but would frequently get stuck in a local maxima with fitness around 0.77, eventually timing out. This suggests that a lack of genetic variety was the culprit, as it's limited what genome crossover can accomplish alone. Increasing the rate to 10% drastically increased convergence rates (as shown in Figure 10, even though the population median stays close to 0.5 for the entire run.

Finally, this implies a one-to-many mapping between reservoirs and readout layers, as there are multiple compatible reservoirs for each previously trained readout layer. This makes the potential use of a smaller generative genome for evolving RRC systems interesting. Even though it hits fewer points in the RBN fitness landscape than the fixed genome used in this paper, a large amount of these points are still usable for each instance of a working readout layer.

VII. Future work

A. Required Complexity of Tasks

Only RBN Reservoirs of size $N = 100$ are looked at in this paper, and in [14] reservoirs of size $N = 500$ are used. Neither might be the optimal size for an RBN, with the best dynamics and problem accuracies potentially obtained at different reservoir sizes. If it turns out that a given task can be solved just as easily for a reservoir of size $N = 50$ as $N = 500$ one might as well use the smaller reservoir, saving bits, hardware, and the environment in one go. If one can obtain the relationship betwen how complex a reservoir has to be to solve a given task, one could predict i.e. how large the water bucket reservoir presented in [3] actually has to be.

B. Generative genomes

A fixed-representation genome was used in this paper to evolve functionally equivalent reservoirs, with the fitness function requiring the reservoirs to adapt to a given readout layer. Alternatively one could evolve RBNs towards the dynamics known to be useful for computation. This can be done with a generative genome, where the genome defines how the graph grows to its adult form, as opposed to describing the edges and nodes of the adult RBN directly.

C. Scaling up Simulation

Software simulations of RBNs and the training of the corresponding RRC systems can be rather slow. Therefore it might be more efficient to implement the RRC system in an FPGA, or as an accellerated task on a supercomputer or graphics card.

VIII. Conclusion

A functioning RBN Reservoir Computing system was implemented, and its results validated against and found in accordance with those from a previous publication: A positive correlation between the computational capability of a reservoir and its actual performance is found. The optimal connectivity for homogenous reservoirs is found to be $K = 3$ as opposed to $\langle K \rangle = 2$ for heterogenous reservoirs. Finally, the required input connectivity is found to rise with the presence of chaotic dynamics in the reservoir.

A genetic algorithm is created for evolving functionally equivalent reservoirs for use under the same readout layer in RRC systems. There turns out to be a great number of functionally equivalent reservoirs, and finding them is easy and efficient. A many-to-one mapping between reservoirs and allready trained readout layers is therefore present. Their dynamics and properties are however more representative of the general reservoir population than the original reservoir. This makes the potential use of a smaller generative genome for evolving RRC systems interesting. Even though it hits fewer points in the RBN fitness landscape than the fixed genome used in this paper, a large amount of these points are still usable for each instance of a working readout layer.

The use of the relatively simple Random Boolean Network in Reservoir Computing is a relatively new approach, but one found fruitful. The implemented RRC and GA systems are generic and reusable and verified to work according to specification, opening up for further research within the field during the authors master's thesis.

References

[1] 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.

[2] Sigve Sebastian Farstad. "Evolving Cellular Automata in-Materio". In: *Unpublished* (2015).

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

[4] *Forprosjekt code repository on GitHub*. https://github.com/aleksanb/RBN-Reservoir-Code.

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

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

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

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

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] David Verstraeten et al. “Oger: modular learning architectures for large-scale sequential processing”. In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 2995–2998.

[16] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), pp. 67–82.

[17] Tiziano Zito et al. “Modular toolkit for Data Processing (MDP): a Python data processing framework”. In: *Frontiers in neuroinformatics* 2 (2008).