Investigating Rules and Parameters of Reservoir Computing with Elementary Cellular Automata, with a Criticism of Rule 90 and the Five-Bit Memory Benchmark

Tom Eivind Glover[1,4]
Pedro Lind[1]
Anis Yazidi[1]
Evgeny Osipov[2]
Stefano Nichele[1,3]

[1] *Department of Computer Science*
Oslo Metropolitan University
Oslo, Norway

[2] *Department of Computer Science, Electrical and Space Engineering*
Luleå University of Technology
Luleå, Sweden

[3] *Department of Computer Science and Communication*
Østfold University College
Halden, Norway

[4] *tomglove@oslomet.no, tom.eivind.glover@gmail.com*

Reservoir computing with cellular automata (ReCAs) is a promising concept by virtue of its potential for effective hardware implementation. In this paper, we explore elementary cellular automata rules in the context of ReCAs and the 5-bit memory benchmark. We combine elementary cellular automaton theory with our results and use them to identify and explain some of the patterns found. Furthermore, we use these findings to expose weaknesses in the 5-bit memory benchmark as it is typically applied in ReCAs, such as pointing out what features it selects for or solving it using random vectors. We look deeply into previously successful rules in ReCAs such as rule 90 and explain some of the consequences of its additive properties as well as the correlation between grid size and performance. Additionally, we present results from exhaustively exploring ReCAs on key parameters such as distractor period, iterations and grid size. The findings of this paper should motivate the ReCAs community to move away from using the 5-bit memory benchmark as it is being applied today.

Keywords: Reservoir Computing; Cellular Automata; Reservoir Computing with Cellular Automata (ReCAs); Edge of Chaos

https://doi.org/10.25088/ComplexSystems.32.3.309

1. Introduction

A common method in the machine learning (ML) community is to rely on cloud computing, cluster computing or even supercomputers for training. These designs perform well due to their accuracy and flexibility; however, training requires large amounts of computation, and computation requires electricity, and electricity has a cost. This cost is not only money but can also be put into context as an environmental burden [1].

A common but energy-intensive paradigm for artificial intelligence (AI) is to use deep learning with error backpropagation. Furthermore, some data, such as time series data, typically requires some form of recurrent neural network (RNN), which is often trained using backpropagation through time. These methods not only require a good deal of energy and time to train, but they also require careful design of the network architecture as well as global error calculations, which imply a form of centralized control.

Reservoir computing (RC) is a framework that allows for the speeding up of the training by relying on an untrained but dynamically set up substrate. The high-dimensional results from the substrate are then separated by a trained single linear output layer. Relying on only a single trained layer would naturally require less training time. The concept originated in echo state networks [2] and liquid state machines (LSMs) [3] and since then has been commonly referred to under the umbrella term RC. One interesting feature of RC is that it is very substrate independent and can utilize many different kinds of substrates [4].

One such substrate is cellular automata (CAs). We can consider CAs a special case of neural networks with discrete states and uniform connectivity. One advantage of this substrate is that it can be implemented into hardware such as field-programmable gate arrays (FPGAs) [5]. This hardware implementation affords energy-efficient computation and fast execution, as the FPGAs can support parallel execution in a circuit-level performance. Furthermore, in some contexts like smart devices or drones (Edge AI), the energy-intensive and/or cloud-based solutions are not feasible due to availability, latency, energy usage or even privacy [6]. Some hyperdimensional computing (HDC) systems such as [7] show promising results and include a cellular automaton (CA) in FPGAs serving for one part of the HDC architecture.

Reservoir computing with cellular automata (ReCAs) is the combination of the two, RC with a CA for the reservoir substrate. It was first explored in [8] and subsequently in many other studies [9–14]. All these studies are mostly exploratory and none uses the entire rule

space of elementary CAs (ECAs). In this paper, we investigate and elucidate interesting results in ReCAs, such as rule 90's relationship to grid size. Additionally, we expand the results of our previous work [15] by testing many additional standard parameters on the x-bit memory benchmark. The main findings of this paper are:

- We show an example of rule 90 being capable of changing fundamental behavior based on small changes in a parameter. Additionally, we show consequences of rule 90's additive behavior and tie it into ReCAs. Furthermore, we show that its performance on the x-bit memory benchmark does not necessarily make it a good reservoir.
- We demonstrate how deterministic chaos and sufficient dimensions can be used to solve the x-bit memory benchmark and argue why it is not necessarily a good thing.
- We identify and discuss some weaknesses in the x-bit memory benchmark as it is commonly applied today.
- We extend past work with additional extensive results in ReCAs on the x-bit memory benchmark on the parameters distractor period D_p, iterations between input I and D_p and grid size combined.

2. Background

2.1 Cellular Automata

CAs are a simple model consisting of a grid of cells possessing a limited set of k discrete states placed on a uniformly connected grid, typically in one or two dimensions. The cell state changes iteratively, depending on the state of the neighbors. Which neighbor state combination results in which next state is determined by a lookup table, typically called the transition table (TT). CAs were first used to study self-replication by John von Neumann in 1940, but not published before 1966 [16]. It can be considered an idealized system for parallel and decentralized computation [17].

2.1.1 Elementary Cellular Automata

The simplest cellular automaton (CA) form is an elementary CA (ECA), a subset of CAs with two discrete states, arranged in one dimension, and a neighborhood schema of direct left and right neighbor as well as itself. Contained within this rule space of just $2^{2^3} = 256$ rules is a universe that can be extensively studied. Each individual rule is conventionally named after the output of the

https://doi.org/10.25088/ComplexSystems.32.3.309

transition table in decimal; for example, Binary(01 101 110) = Decimal(110). Rule 110 has even been shown to be computationally universal [18], but we can question whether that is a useful definition of computation for a parallel and distributed computational substrate [19].

2.1.2 Additive Cellular Automata

All ECA rules are equivalent to simple formulas in algebraic or Boolean logic. As an example, rule 90 (Figure 1), consists in an XOR between the right and left neighbor. The additive CAs (ACAs) are a special group of ECAs that can be simplified to adding the relevant neighbors together and applying modulo 2 to the result. The list of ECAs that can be reduced to simple additive operations based on the neighborhood is shown in Table 1. This additive property allows for a more complete understanding of the many global properties of the additive CA (ACA) rules [20]. A more accessible explanation of ACAs can be found at [21].

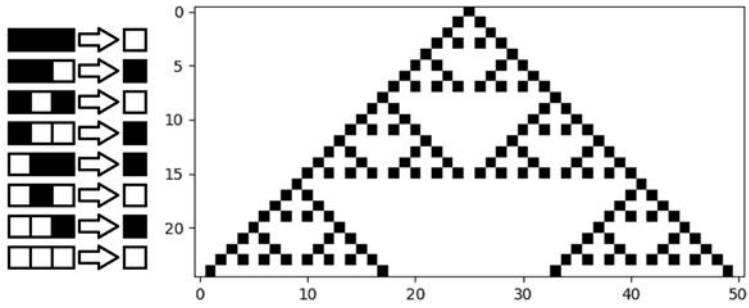

Figure 1. The first 25 steps of rule 90 with a single central on state as initial condition.

2.1.3 Differentiation Pattern

One tool that is sometimes used to analyze CAs is the CA difference pattern. The process is simple: take a CA, flip a single bit in the initial condition, compare to the original and see what is different in the development. In Figure 2 we see this in rule 110; for this rule the difference pattern will vary quite a bit based on the initial condition. In Figure 3 we see the difference pattern for rule 90. Note that the difference pattern is exactly like the standard central pattern generated by rule 90, as can be seen in Figure 1; this is a feature of the additive CA.

Rule	Boolean
Rule 0	0
Rule 60	$c_{-1} \oplus c_0$
Rule 90	$c_{-1} \oplus c_1$
Rule 102	$c_0 \oplus c_1$
Rule 150	$c_{-1} \oplus c_0 \oplus c_1$
Rule 170	c_1
Rule 204	c_0
Rule 240	c_{-1}

Table 1. ACA and corresponding Boolean form. In this notation, c_{-1}, c_0 and c_1 are the left, central and right neighbor, respectively.

Figure 2. Rule 110 difference pattern. Left to right, random initialization, centered bit changed and difference pattern.

Figure 3. Rule 90 difference pattern. Left to right, random initialization, centered bit changed and difference pattern.

2.1.4 Equivalent Elementary Cellular Automata

Many ECA rules are equivalent to other rules under simple mirror and complement transformation. The mirror transformation is the reflection of the rule over the x axis (e.g., rule 170 is left-shift, and its mirror is rule 240, right-shift). The complement transformation is the flipping of all bits in the TT, both neighborhood and result, then reordering (e.g., rule 0 is everything turns to white; complement is rule 255, everything turns to black). In addition, there is the combination of both mirror and complement transformations. Using these equivalent classes, the 256 ECA rules get reduced to 88 unique rule groups, Table 2. Typically, it is not necessary to evaluate all the rules within a group but rather to use the minimum equivalent (ME) rule to

represent its group. The 88 ME rules can be considered to be the representation of the entire rule space, though to be clear, the equivalent under the transformations does not mean they are fully equal. If the computation that is required is a right-shift, rule 240 would be the most efficient way to achieve this and would outperform rule 170.

Rule	Equivalent	Rule	Equivalent	Rule	Equivalent
0	255	35	49,59,115	108	201
1	127	36	219	110	124,137,193
2	16,191,247	37	91	122	161
3	17,63,119	38	52,155,211	126	129
4	223	40	96,235,249	128	254
5	95	41	97,107,121	130	144,190,246
6	20,159,215	42	112,171,241	132	222
7	21,31,87	43	113	134	148,158,214
8	64,239,253	44	100,203,217	136	192,238,252
9	65,111,125	45	75,89,101	138	174,208,244
10	80,175,245	46	116,139,209	140	196,206,220
11	47,81,117	50	179	142	212
12	68,207,221	51		146	182
13	69,79,93	54	147	150	
14	84,143,213	56	98,185,227	152	188,194,230
15	85	57	99	154	166,180,210
18	183	58	114,163,177	156	198
19	55	60	102,153,195	160	250
22	151	62	118,131,145	162	176,186,242
23		72	237	164	218
24	66,189,231	73	109	168	224,234,248
25	61,67,103	74	88,173,229	170	240
26	82,167,181	76	205	172	202,216,228
27	39,53,83	77		178	
28	70,157,199	78	92,141,197	184	226
29	71	90	165	200	236
30	86,135,149	94	133	204	
32	251	104	233	232	
33	123	105			
34	48,187,243	106	120,169,225		

Table 2. The group of equivalent rules. The primary column is the minimum equivalence rule that represents its group of equivalent rules, if any.

2.1.5 Cellular Automata Classifications

A favorite pastime of the CA community is to classify and group CAs into discrete groups. In this paper, two classifications will be explained, Wolfram classification and ECAs with memory (ECAM). There are many other classifications, such as Li and Packard's classification, index complexity classification or power spectral classification. A good overview of these previously mentioned classifications and many more can be found in [22].

2.1.6 Wolfram Classification

One well-known classification is the Wolfram classification [23], where CAs with random initial conditions were explored. It was observed that the distinct rules for most initial conditions fit within one of four classes:

- *Class 1*. Evolves into homogenized states.
- *Class 2*. Evolves into a set of simple separate stable or short periodic structures.
- *Class 3*. Evolves into a chaotic pattern.
- *Class 4*. Evolves into complex localized structures that sometimes are long lived.

Examples of the four behaviors can be seen in Figure 4.

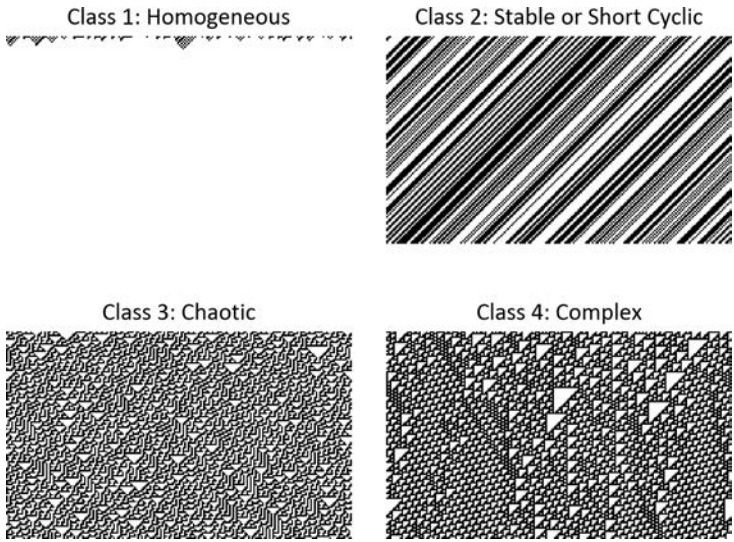

Figure 4. Examples of the four Wolfram classes. Top: rules 40 and 170, bottom: rules 30 and 110. All rules with random initialization, grid size 200 and 120 CA steps.

2.1.7 Elementary Cellular Automata with Memory

ECAM is another classification of ECAs. In this classification, ECAs are paired with a memory function that combined the n previous cell states using minority, majority and parity function [24]. The ECA rules were then classified depending on how the rule changed behavior or not in relation to the Wolfram classification. The ECA rules were found to group into three classes:

- *Strong*. Most memory functions change the rule to another different class quickly.
- *Moderate*. Memory functions can transform to a different class and conserve the same class as well.
- *Weak*. Memory functions are unable to transform into another class.

Note that these three definitions are slightly switched in comparison to the original publication: here the definition for strong and weak have been mutually exchanged. The reason for this is that after reviewing the evidence of the same paper, there is a contradiction of the examples and evidence. For example, rule 126 shows complex behavior under some memory functions. Additionally, consider that three prepositions of the paper in relation to the classification definitions contradict each other in the original work. This correction does not alter the distribution of the rules. As such, there is no reason to believe that this is a mistake made in bad faith, as it does not seem to change the quality of the results. Yet, it is still important to clarify, as building our rationale on the conclusion of particular rules from a particular class would become quite wrong if applied under the previous definitions.

The ECA rules in regard to ECAM and the Wolfram classification can be found in Table 3.

Class I	Rules
Strong	128
Moderate	8,32,40,136,160,168
Weak	0

Class II	Rules
Strong	2, 7, 9, 10, 11, 15, 24, 25, 26, 34, 35, 42, 46, 56, 57, 58, 62, 94, 108, 130, 138, 152, 154, 162, 170, 178, 184
Moderate	1, 3, 4, 5, 6, 13, 14, 27, 28, 29, 33, 37, 38, 43, 44, 72, 73, 74, 77, 78, 104, 132, 134, 140, 142, 156, 164, 172
Weak	12, 19, 23, 36, 50, 51, 76, 200, 204, 232

Class III	Rules
Strong	18, 22, 30, 45, 122, 126, 146
Moderate	
Weak	60, 90, 105, 150

Class IV	Rules
Strong	41, 54, 106, 110
Moderate	
Weak	

Table 3. All minimum equivalent ECAM classifications.

2.2 Edge of Chaos and λ Parameter

The parameters space of a complex system often has a phase transition between order and disorder; this phase transition region is often called "edge of chaos." It is theorized that this region commonly contains the highest capacity for computation, defined as transformation, manipulation and storage of information.

Langton [25] explored this theory in one-dimensional multistate CAs with enlarged neighborhoods and found that the CA rule space forms a phase transition between order and chaos when organized over a λ (lambda) parameter. The λ parameter starts by defining a state as the quiescent state. To generate transition tables with a given λ value, we can allocate to each TT entry a random number α uniformly distributed between 0 and 1 and attribute the quiescent state to all entries with $\alpha < \lambda$ and a nonquiescent state to the others. Using this method, Langton generated different candidate rules in several regions of the rule space over the λ parameter. He showed that the rule space organizes into a phase transition between order and chaos and that strong candidates for computation are more likely to be found there. Notably, this λ method does not seem to work in the ECA rule space, as mentioned in [25] and previous work.

2.3 Kolmogorov Complexity, Lossless Compression, Shannon Entropy and Block Decomposition

One rigorous way to assess behavior in ECAs can be found in [26]. In this paper, Zenil explored CAs, asymptotic behavior and complexity. First, this paper demonstrates how rule 22 can behave differently depending on the initial condition and how this is a problem of the Wolfram classification. Then the paper alleviates this problem by redefining the Wolfram classification as the limit of the rule's behaviors, for example, to the highest class (complexity) of all its behaviors. Second, the paper then explores ECAs and estimates the Kolmogorov complexity [27] using three methods: lossless compression, Shannon block entropy and block decomposition. Using these metrics and the

new classification method, some ECA rules are reclassified. Finally, using these methods, the paper estimates the complexity ratio of different rule spaces.

2.4 Reservoir Computing

RC is a substrate-independent framework for computing. RC is independent because it works on many different substrates, but to be clear, different substrates would of course have different capabilities. The RC framework consists of three parts: the input, the untrained reservoir and the output.

The input part encodes some information into the untrained reservoir and typically into higher dimensions. The untrained reservoir typically expands, modifies or changes the information, but could in the context of the framework be considered a black box, as seen in Figure 5. The output part is typically linear and does dimensional reduction and extracts useful features.

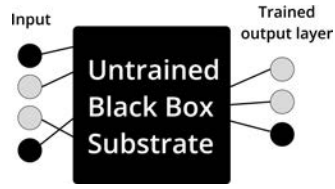

Figure 5. RC as a substrate-independent framework.

The RC concept originated in echo state networks (ESNs) using recurrent neural networks (RNNs) as a substrate [2] (see Figure 6) and in LSMs using a spiking neural network for a substrate [3]. Since then, both ESNs and LSMs and a host of other substrates have been put under the umbrella term of RC. Due to RC's substrate-independent nature, many different substrates have been explored and/or compared [4]. Some explore different configurations of topology as

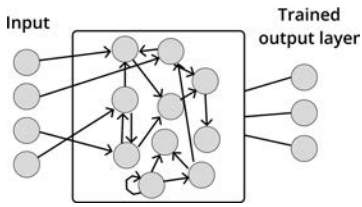

Figure 6. Basic network architecture of an echo state network (ESN).

in [28], where instead of the typical one big reservoir, deep layered sub reservoirs were analyzed. RC is also a very popular method with physical reservoirs [4]; as an extreme example, in [29] it was demonstrated that RC can use the surface waves on a bucket of water as a reservoir and they successfully solved speech recognition and XOR tasks using this substrate. One very interesting substrate is real biological neural networks (BNNs), specifically disassociated neurons that self-organize over a microelectronic array [30].

There is also evidence of RC being a useful trick for computation (one of many) used in biology. In [31] they showed that a linear classifier can extract information about the short-term past stimulus (images, XOR) from the primary visual cortex of an anaesthetized cat. Also in other biological, but also computational processes, there is some evidence of RC. In [32] the ESN (RC) model was used to simulate an example of a known genetic regulation network (GRN) process and performed satisfactorily. Similarly in [33] a liquid state machine (LSM) RC model was used.

2.4.1 Echo State Property and Fading Memory Property

An important property in ESNs is the echo state property (ESP). To have this property, the reservoir must, given some input signal, asymptotically remove the information of the initial condition. In [2], it is shown that for a reservoir with specified conditions, it violates the ESP if the spectral radius of the weight matrix is larger than 1, and it was empirically observed that for spectral radius below 1, the ESP is given. Note that in [34] Jaeger warns that this does not mean that ESP is granted for any system with a spectral radius of below 1 (asymptotically stable). It is not a necessary or sufficient condition.

Similar to the ESP is the concept of the fading memory property. It states that an input/output system is said to have fading memory when the outputs associated with inputs that are close in the recent past are close, even when those inputs may be very different in the distant past [4, 35].

2.5 Reservoir Computing with Cellular Automata

The first study that introduced CAs as a substrate in RC is [8]. In this study, the Game of Life and several ECA rules were investigated as reservoir substrates and tested on a 5-bit and a 20-bit memory benchmark. In addition, it presents a theoretical comparison of CAs versus ESNs, using the metric of the number of operations needed to solve the benchmark, which documents a clear advantage of using CAs.

As an ECA reservoir only relies on simple discrete binary interactions between cells (e.g., Table 1), it affords a hardware-friendly implementation of a substrate. The problem (perhaps ironically)

https://doi.org/10.25088/ComplexSystems.32.3.309

becomes how to implement the readout layer in hardware. In [5] ReCAs using ECAs with a max-pooling and softmax strategy were implemented on a field-programmable gate array (FPGA). In [36] a CA was implemented on a complementary metal-oxide semiconductor (CMOS) combined with a custom hardware support vector machine (SVM) implemented in resistive random-access memory (ReRAM). In [37] a synthesized hardware implementation of ReCAs using ECAs with a max-pooling and ensemble bloom filter classifier was used, showing impressive results compared to "state of the art" in terms of energy efficiency, memory usage and area (number of gates) usage, but with comparably poor accuracy [5].

Since the first paper on ReCAs [8], other works have studied ReCAs using the 5-bit memory benchmark. In [9], the structure of the CA was changed to a deep-layered architecture and compared to a single layer, which resulted in noticeable performance improvements. In [10] the CA substrate was organized as consisting of two regions of different ECA rules. Different combinations of rules were explored, and some of them showed great promise. In [11] different methods of cell history selection that are used for the classification model were explored on the 5-bit memory task, a temporal order task and arithmetic and logic operation tasks. In [13] CA rules with multiple states and larger neighborhoods were evolved and then tested on the 5-bit memory benchmark. In [38] ECAs and asynchronous ECAs were tested and compared on the 5-bit memory benchmark, mainly in the context of the distractor period. In [15] the full ECA set was tested using key parameters of number of bits N_b, redundancies R and grid size.

ReCAs are also used on benchmarks other than the 5-bit memory benchmark. For example, in [5, 37] ReCAs were implemented in hardware and tested using the Modified National Institute of Standards and Technology database (MNIST). In [14] ReCAs were used to solve tasks of sine and square wave classification, nonlinear channel equalization, Santa Fe laser data and iris classification.

3. Methodology

3.1 5-bit Memory Benchmark

The 5-bit memory benchmark traces its root to the short long-term memory task introduced in [39]. It is often cited as the source [8–10, 13], but none of the benchmarks in [39] are the 5-bit memory benchmark, although some of them are very similar in intention. The earliest source where the 5-bit memory benchmark is recognizable is in [40], but named "noiseless memorization," corroborated with the clearer and more detailed explanation of the benchmark in [41, p. 47] and in [42].

The 5-bit memory benchmark's goal is to test whether a system is capable of memorizing a 5-bit string and reproducing it at a later stage. An example of the 5-bit memory benchmark can be seen in Table 4. The benchmark has four input channels, where only a single channel can be active at a given time. The first two input channels are dedicated to the five bits. The bits are fed into the system sequentially over five steps. The first input channel can be viewed as the "pure" five bits and the second as the reversed five bits. The third input channel is dedicated to constantly feeding input into the system during the distractor period and the output stage. The fourth input channel is dedicated to the cue signal, signaling that the output is to be given. The benchmark has three output channels, where one and only one should be active at a given time. Note that some earlier examples have four output channels but one is dropped, as it is never intended to give output. The first two are dedicated to the original five bits inserted into the system and should sequentially output them following the cue signal; the final output channel should give a signal in all other cases. Due to the nature of this output, we can abstract and view the task as a temporal classification problem.

Step	Input				Output			Stage
1	1	0	0	0	0	0	1	Input bits to memorize
2	1	0	0	0	0	0	1	
3	0	1	0	0	0	0	1	
4	0	1	0	0	0	0	1	
5	1	0	0	0	0	0	1	
6	0	0	1	0	0	0	1	Distractor period
...	0	0	1	0	0	0	1	
204	0	0	1	0	0	0	1	
205	0	0	0	1	0	0	1	Cue signal
206	0	0	1	0	1	0	0	Output bits to memorize
207	0	0	1	0	1	0	0	
208	0	0	1	0	0	1	0	
209	0	0	1	0	0	1	0	
210	0	0	1	0	1	0	0	

Table 4. Example of the 5-bit memory task with distractor period of 200 and input of the number 25 in binary form. Artifact inspired by [13].

In this paper, we will often call it the x-bit memory benchmark, as we have varied the number of bits to be memorized. Also, note that the 20-bit memory benchmark in at least some of the previous sources is not the same as the 5-bit memory benchmark but with 20 bits to memorize. The 20-bit memory benchmark uses seven input channels, five for the input and a bit length of 10.

https://doi.org/10.25088/ComplexSystems.32.3.309

3.2 X-bit Memory Task in Reservoir Computing with Cellular Automata

The benchmark itself has been presented independently of the chosen substrate. We now detail the specific benchmark definition in the context of a CA substrate.

As shown in Figure 7, additional steps have been added. First, the encoding part, which considers how to inject the input into the substrate. For binary data, there have been many different methods proposed and/or implemented [8, 9, 11], but there are some common tendencies. The input is usually randomly mapped into the CA in several redundant mappings given by the redundancy R parameter. This random mapping stays fixed throughout the experiment. In [8], it is not completely clear, but it is claimed to be done using a vector of the same size as the input. Considering Figure 4 and Table 3 from said paper, there is a conflict, as that would mean that for rule 90, $T_0 = D_p = 160$ is more difficult than $T_0 = D_p = 200$. This is entirely possible but violates a claim in the paper that "there is a polynomial increase in the minimum required reservoir size with number of bits to be remembered and but a logarithmic increase with distractor period." Regardless, in later experiments [9, 10] a larger vector is used. The size of this new vector is given by the L_d parameter.

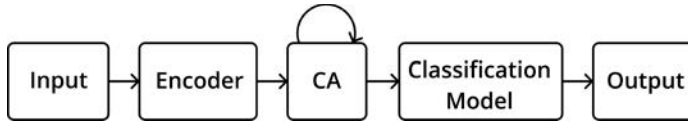

Figure 7. Simple model of the different parts of ReCAs using the x-bit memory benchmark.

The simplest way to encode the input into the CA is to overwrite the current state of a mapping. Since this might overwrite important information in the CA, other methods have been used, such as simple binary operations between the current state of the CA and the new input value. In this paper, we use XOR between the current state and input to encode into the CA.

The second component is the actual substrate itself, a CA in the paper herein. This operates like any CA and can be of any dimensional ity, with any number of states and with any neighborhood scheme. Commonly, the CA permutes the input several steps before giving the next input. Therefore, we can say that the CA and x-bit memory benchmark exist on different timescales, as several CA steps are performed between I/O steps. The number of additional CA steps before the next input is represented by I. Some examples of what the CA history of the reservoir can look like are in Figures 8 and 9.

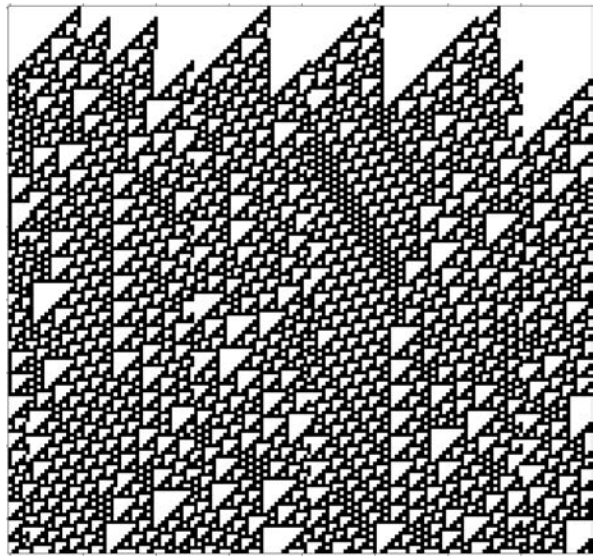

Figure 8. Rule 110, input 10110, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

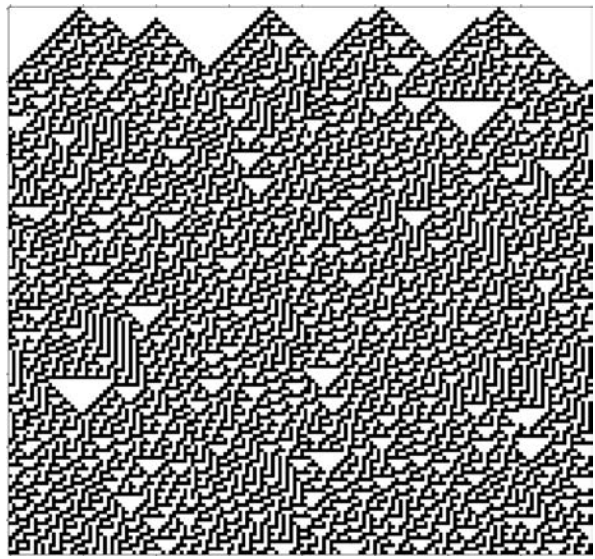

Figure 9. Rule 30, input 10110, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

The third part is the classification model (sometimes known as the decoding stage). This stage uses the state of the CA (or a set of previous states) to classify and produce an output. Due to the nature of

https://doi.org/10.25088/ComplexSystems.32.3.309

RC, the classification model is commonly a linear model. Consider that if we were to use a deep neural network as a classification model rather than a linear one, this would cast doubt about whether the separation was done in the model or the substrate. Therefore, using a linear model demonstrates that the system is doing RC. When limited by this requirement, two common models are typically used, that is, linear regression and SVM with a linear kernel. If linear regression is used, it would need to be paired with a rounding function or some max confidence method to produce clear output. If SVM is used, we can take advantage of the fact that the output is always only a single class and can therefore handle the model as a standard SVM model.

3.3 Perfect Score Metric versus Weighted Average Metric

In this paper, two different metrics were used to compare rules. The perfect score P metric is the traditional and commonly used metric for scoring on the x-bit memory benchmark. In this metric, every classification for every permutation has to be correct; if a single classification is wrong, no credit is given. In the example, Table 4 has 210 classifications. Therefore, if one of the total $210 \times 32 = 6720$ (32 different permutations of 5 bits) is incorrect, a failed score of 0 is given. Such a metric is very strict and makes it hard to identify configurations close to solving the problem; therefore, we also use the weighted average $\overline{W}$ metric. The average is weighted to adjust for the fact that a large set of the classifications in the x-bit benchmark has the same output. In the example in Table 4, 205 out of the 210 classifications are the "no output state." Therefore, correctly classifying 205 out of the 210 states is trivial and could be done even without a reservoir or even without input. The ratio of trivially classifiable states depends on two parameters, the number of bits N_b and distractor period D_p, and can be found using equation (1):

$$W = \frac{D_p + (N_b)}{D_p + (2 \times N_b)}. \tag{1}$$

Given that the fraction of correctly classified states is $\overline{S}_c$, the $\overline{W}$ can then be found using equation (2):

$$\overline{W} = \frac{\overline{S}_c - W}{1 - W}. \tag{2}$$

Both metrics are then made into percentages for ease of comparison.

4. Experimental Setup

4.1 Libraries and Source Code

The CA reservoir was modeled using the EvoDynamics framework [43]. EvoDynamics natively supports CA reservoirs built from TensorFlow, enabling the CA to run on TensorFlow-GPU. In addition, SkiKit-learn [44] was used to create the SVM with a linear kernel, used for classification. The source code for the experiments can be found at [45], and a more accessible way to navigate the data generated and the graphs generated can also be found on GitHub at [46].

4.2 Rule Order

In the following sections, the rule space is sorted in the same way as [15]. The intention is to organize the rule space such that rules with similar behavior are placed adjacent. The rule space is ordered in descending order of importance from three factors, first using Wolfram classification class 1, 2, 3, 4; then on ECAM strong, moderate and weak; and finally, the tie breaker is performance on $N_b = 3$ in [15]. Wolfram classification was picked because it is the de facto standard classification and more importantly because it correlates well with behavior. ECAM was picked because the behavior (specifically classes 3 and 4) correlates well. Note that ECAM and the x-bit memory benchmark both share the word memory but they refer to different concepts [24]. Indeed, memory refers to combining previous CA steps with an operation, while in the x-bit memory benchmark, memory refers to recovering earlier input. Finally, $N_b = 3$ was picked because it was seen as the simplest version of this benchmark run so far. We acknowledge that this unconventional order might make it difficult in comparison to numeric sorting, or if you want to find your favorite rule in the diagram.

Only the ME rule of each group is presented in the diagram, and in addition, rules that did not perform beyond the trivial case of performing 0 on the $\overline{w}$ on any experiment are not shown in the rule diagrams. This is in slight contrast to [15], in that work rules were filtered if they had a trivial performance on individual experiments for the sake of space usage, but this makes it harder to navigate the many different versions of the diagram; therefore, this is avoided in this paper. No Wolfram class 1 made this distinction of nontrivial performance; this is not surprising but might be worth noting to avoid confusion.

4.3 Overview of Explored Reservoir Computing with Cellular Automata Parameters

This benchmark and the CA setup have many parameters; they have been introduced on demand, and to establish a better overview, we list them here.

https://doi.org/10.25088/ComplexSystems.32.3.309

- R is the number of regions the CA is divided into.
- L_d is the width of the individual R.
- I is the number of CA-only iterations between input and historical states (height) of the CA the SVM has access to.
- D_p is the length of the distractor period, or the number of distractor inputs between the last memory input bit and the cue signal.
- N_b is the number of bits to be memorized.

R, I and L_d can be seen in Figure 10, and both D_p and N_b can be seen in Table 4. Note that there are many parameters of the benchmark and CA setup that are not explored in this paper, for example, encoding strategies.

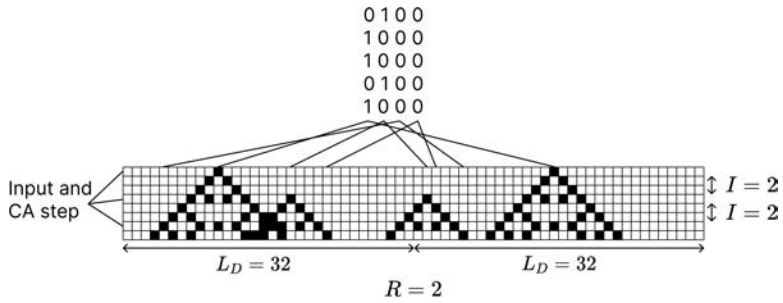

Figure 10. Example showing R, I and L_d. Additionally, the top stream is an example of how input is encoded temporarily into the reservoir.

4.4 Encoding Strategy

The 5-bit memory benchmark is a temporal benchmark, meaning the input streams are inputted into the CA over time. The past input and their "echos" can therefore still affect the CA configuration, and the CA is no "tabula rasa" of only quiescent cells. In these experiments, given the previous state S, the input streams I_s are XORed together $S \oplus I_s$. A more general encoding strategy that would work for more than two states would be: given the number of states N_s; $(S + I_s)\%N_s$. This would be equivalent to the XOR strategy for binary states.

5. Results

We present our results in this section, starting with the quantitative results, as they do build up some evidence for our in-depth results, which are discussed later.

5.1 Exploring the Parameter Space

The quantitative experiments run in this paper greatly extend what was done in [15], and this paper can be seen as a completion of important hyperparameters, as it extends into parameters not explored in the previous paper. We will go through three quantitative experiments, one exploring the landscape of I, one exploring D_p and finally one exploring D_p and L_d in combination. In all experiments, any unspecified variable uses the default of $R = 4$, $L_d = 40$, $I = 2$, $D_p = 200$ and $N_b = 5$. All results are out of at least 100 runs; some are more, as past experiments are also used in the results—for example, in [15] experiments are made over the L_d parameter in the same parameter space as this paper's D_p and L_d experiment, and that crossover region would then have results out of 200 runs. To be clear, this was not done in some attempt to P-hack, but simply because the past experimented parameter values were not excluded from later experiments. Finally, we will also look at qualitative spacetime diagrams of some ReCAs.

5.2 *I* Experiment

The I parameter results are shown in Figure 11. It is important to note that this parameter controls both reservoir height and the number of pure CA steps between inputs. The parameter directly affects how many data points get fed into the SVM; therefore, we could assume that any dynamical behavior would be drowned out by this feature, yet what we see is that many rules show clear dynamical behavior. Extreme examples are the rules 3, 56 and 15, but also previously proposed good rules in ReCAs like rules 60 and 90. From the aforementioned rules, rules 15 and 56 show odd I number preference, and rules 3, 60 and 90 favor even-numbered I. In fact, for $I = 3$, rule 150 has a better score than rule 90. If past experiments had

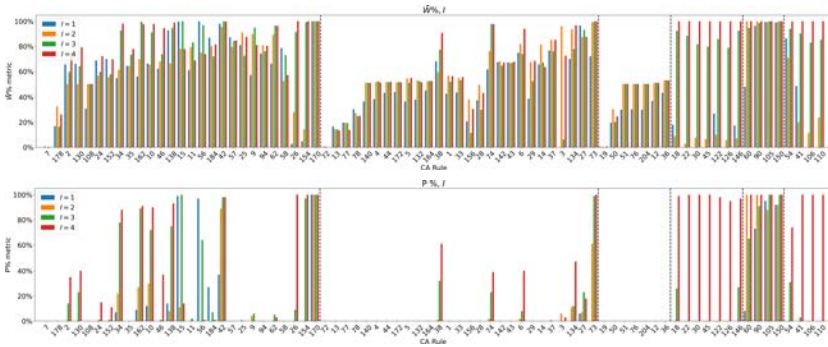

Figure 11. I experiment.

instead of doubling R and I values used odd numbers for parameters or maybe a prime number strategy, a different "best" rule might well have emerged.

5.3 D_p Experiment

The main results are shown in Figure 12. This parameter affects the experiment in two important ways: distractor steps inputted into the system, which would also directly influence the total number of CA iterations to run. This experiment is perhaps the most indicative of rules exhibiting something analogous to fading memory, or the closest thing to fading memory the discrete system of a CA can have. Many rules show that the task becomes harder the larger the D_p, most notably, in the $\overline{W}$ experiment this can be seen in the non-weak class of CA. The weak chaotic rules do seem to exhibit this feature in the P metric as a general trend, but also in detail show very opposed behavior, rule 60 finding $D_p = 100$ harder than $D_p = 200$ or the rules 90, 105 and 150 finding $D_p = 100$ easier than $D_p = 50$.

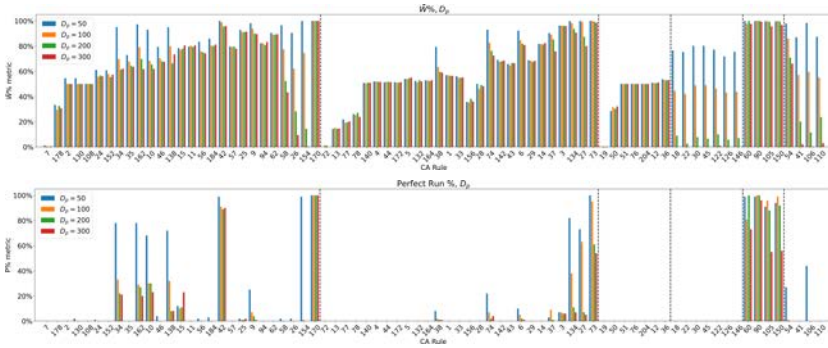

Figure 12. D_p experiment.

5.3.1 $D_p = 4000$

A further longer experiment testing $D_p = 4000$ on rules that got a significant score on $D_p = 300$ is given in Figures 13 and 14. In these results, many of the rules do not seem to find the problem any harder; in fact, the rules 162, 10, 105 and 150 find it easier than $D_p = 300$. Naturally, this does not mean that we are observing a gaining memory, the opposite of a fading memory. First, due to that, the distractor period perturbs at the same locations in a very ordered way, and the CA can still settle into a short periodic attractor despite being continuously perturbed. This can be seen happening in rule 170, due to it looping around the CA, and the distractor input simply cancels the previous distractor input and adds it again in turn [47]. We will go

into this further in the spacetime diagram section. This feature does of course not make the problem easier though, but it can prevent it from becoming harder the longer the D_p is. Second, we know that many rules expand and contract within the CA and that the perturbations can cause different paths in the state. We hypothesize that all or some of the six last I/O steps (i.e., cue signal and return of the original 5-bits steps) will be more separable for $D_p = 4000$ than for $D_p = 3000$. These rules are likely to be in an attractor that is not fully dependent on the initial mapping, and the length of the D_p makes the cue signal hit different steps in that attractor that can be easily separable.

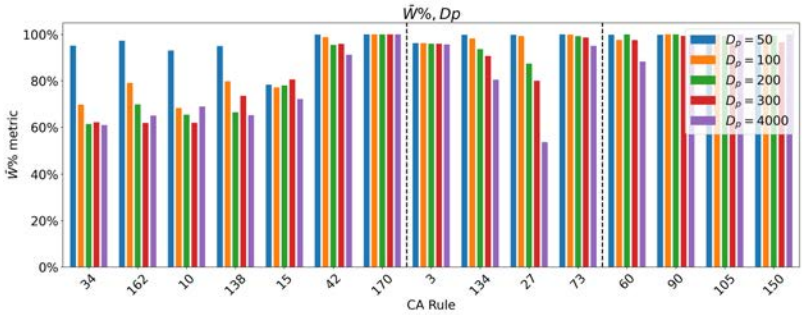

Figure 13. D_p stable experiment, $\overline{W}$ metric.

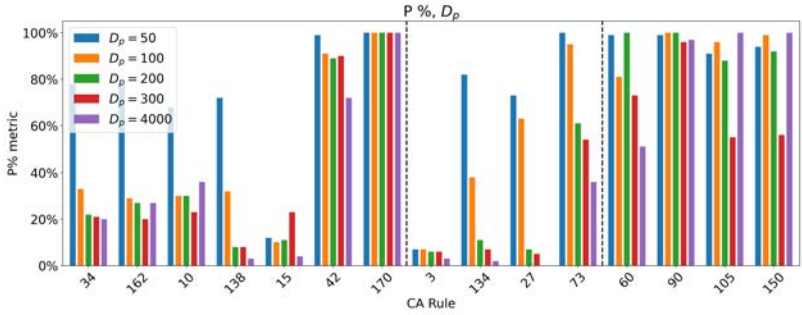

Figure 14. D_p stable experiment, $P\%$ metric.

5.4 D_p over L_d Experiment

In this subsection, we look at results from rules 90, 150, 54 and 170 when looking at how small changes in D_p and L_d affect performance. Rule 90 was picked, as it is viewed as the quintessential rule for ReCAs. Rule 150 was picked as a close contestant to rule 90 that other experiments have shown to have better performance under

specific circumstances, and we hypothesized we would find more examples of this. Rule 54 was picked as it is, within our experiments, the rule within the complex group that had the best performance on small grid sizes. Finally, rule 170 was picked as a comparison, as it is a very simple rule, but also because it performs very well in most circumstances.

5.4.1 Rule 90

In Figures 15 and 16, we observe in rule 90 a clear pattern of good performance in even-numbered L_d, as was found in [15]. In fact, out of the 13 200 trials using odd numbered L_d included in the data for these two diagrams, not a single one got a perfect run, and only for $L_d = 43$ did any configuration get anything close to above the absolute minimum in the $\overline{W}$ metric. The results further strengthen that this combination of parameters $L_d = 40$ and $D_p = 200$ is a very good fit for rule 90 on this benchmark. Also note that there are some examples of specific D_p that have an impact on performance.

Figure 15. Rule 90 $I = 1$.

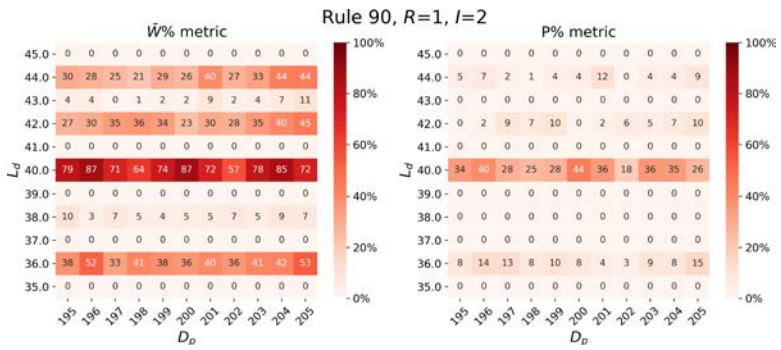

Figure 16. Rule 90 $I = 2$.

5.4.2 Rule 150

For rule 150 in Figures 17 and 18, we also see a similar pattern as in rule 90 of L_d. As for the D_p pattern, it is less clear what the pattern is. Note that for $I = 1$, the best performance was found at $D_p = 198$ and $D_p = 204$, outperforming rule 90 with the same parameters. This again indicates that if different standard parameters were picked for this and past experiments, then perhaps different CA rules would have been inspected and optimized.

Figure 17. Rule 150 $I = 1$.

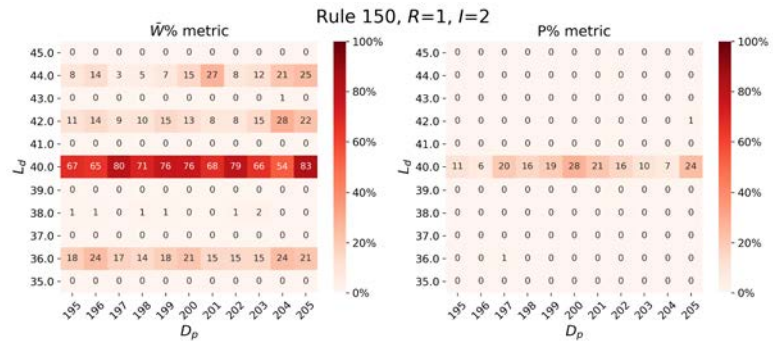

Figure 18. Rule 150 $I = 2$.

5.4.3 Rule 54

We also wanted to inspect a rule classified as complex; rule 54 was picked because it had the best performance on smaller reservoirs in [15]. In Figures 19 and 20 the results are not as dynamic as the additive rules previously inspected, but also there is a weak pattern between the two of odd versus even L_d. In addition, there seems to be a slight "phase change" happening between $L_d = 39$ and $L_d = 40$, most noticeable in $I = 2$.

5.4.4 Rule 170

Finally, we also looked at rule 170 in Figure 21. No dynamical behavior was expected, but there is slight evidence for a preference for $L_d / 3$ for the $P\%$ metric, most notably in $L_d = 36$ and $L_d = 39$. Moreover, this is different in the $\overline{W}$ metric; in fact, the pattern is the opposite: $L_d / 3$ indicates poor performance in this metric.

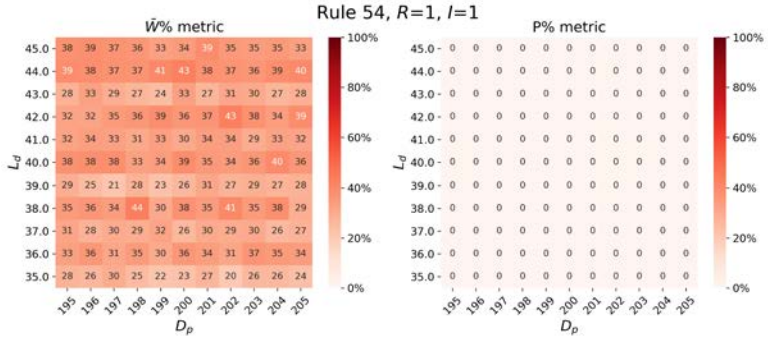

Figure 19. Rule 54 $I = 1$.

Figure 20. Rule 54 $I = 2$.

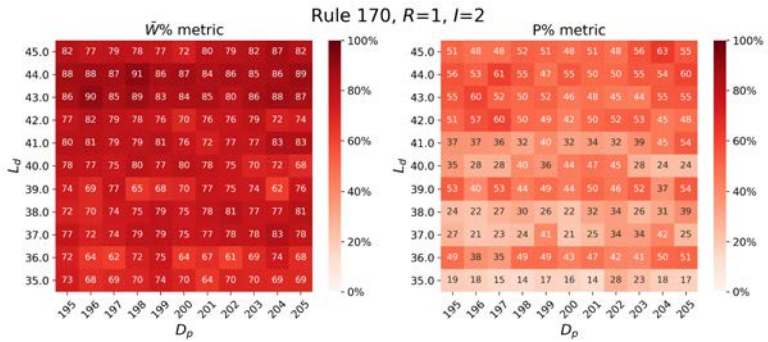

Figure 21. Rule 170 $I = 2$.

5.5 Spacetime Diagrams

In this subsection, we look at some specific spacetime diagrams and rules.

5.5.1 Rule 204

We start with rule 204, a relatively simple rule. Rule 204 is a rule that projects the past state onto the next state. We start with this rule, as it is the simplest rule that allows us to say anything interesting in regard to behavior, but also as a simple way to understand the benchmark and diagram. In Figure 22, we see an example of this rule. The only interference of this downward projection is the information put into it on the same cell: the five bits, the reversed five bits, the distractors and the cue signal—these inputs are all visible in this example. Notably, this rule consistently gets 50% on the $\overline{W}$ metric but never manages to do a perfect run; we can understand this from the diagram. First, this rule can always detect the cue signal, as it is never interfered with due to the downward projection, and the input channels never overlap. Therefore, the SVM will always know whether or not the cue signal has been received. Then the SVM can perfectly separate when it is supposed to return 1 in the "no output" channel (001) from when it is supposed to give a 100 or 010. Yet, this rule can never provide a perfect run because many different inputs will lead to

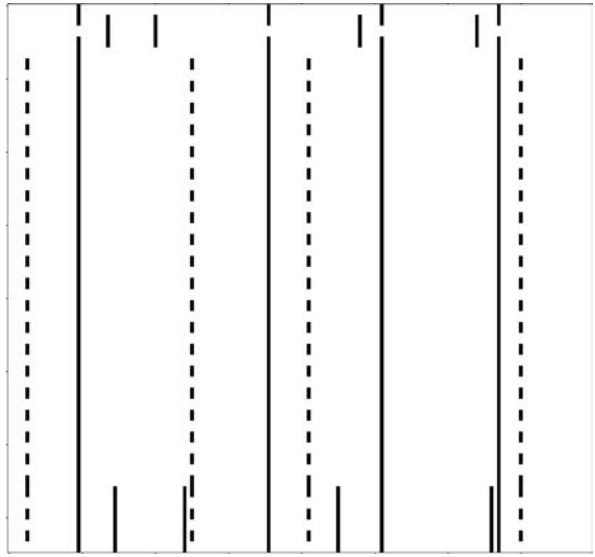

Figure 22. Rule 204, input 10110, $D_p = 40$ with input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

https://doi.org/10.25088/ComplexSystems.32.3.309

the same CA states. All inputs that have an odd number of 1s lead to an on state in that cell, and all inputs that have an even number of 1s lead to an off state in that cell. The SVM cannot separate the CAs that input 10101 from 11100, as they will be the same in the final stage.

5.5.2 Rule 170

Rule 170 is also simple: its behavior is shifting the past state to the left. We can see an example in Figure 23. As mentioned in Section 5.3.1, in a longer run than the example, the distractor period would also loop around and collide with itself again. For the example, the input would take 480 CA steps (160 input steps) after the first distractor input before looping. This period could be potentially shorter with the right input mapping, or longer with a different input frequency but always quite limited. This means that after a certain point, adding more distractor periods would not affect performance. In this rule, the cue signal is also often easy to detect. Having these two features of cue signal detection and direct trajectories of the original input is essentially what makes rule 170 perform so well.

We can take from this that rule 170 demonstrates a weakness of the x-bit memory benchmark. As long as you can separate the cue signal from the other signals and the inputs perturb the system into unique states, all you need to separate is the 2^{N_b} unique input, and with 320 dimensions the SVM would be likely to find one.

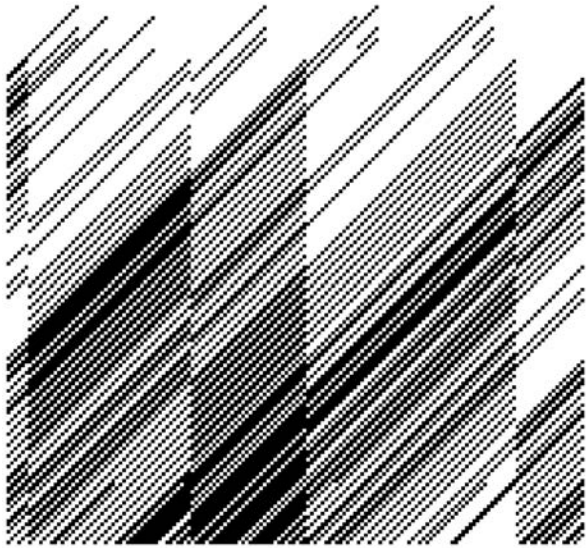

Figure 23. Rule 170, input 10110, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

5.5.3 Rule 54

Rule 54 is seemingly the best-performing complex rule, which inhibits complex abilities of complex trajectories, as described by Wolfram. In Figure 24 we see an example of this, where the interaction of the input creates trajectories that persist, and the regions often called the ether (larger self-similar region) seem to be relatively easy to perturb for the cue signal not to disappear. In Figure 25, we also see that the different input causes different and unique trajectories; these features would not seem ideal, but are still well suited for a memory task like the N-bit memory task.

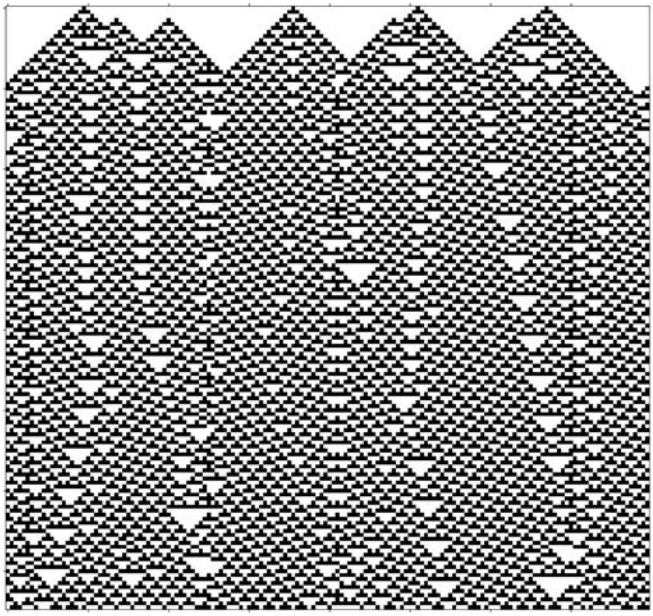

Figure 24. Rule 54, input 10110, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

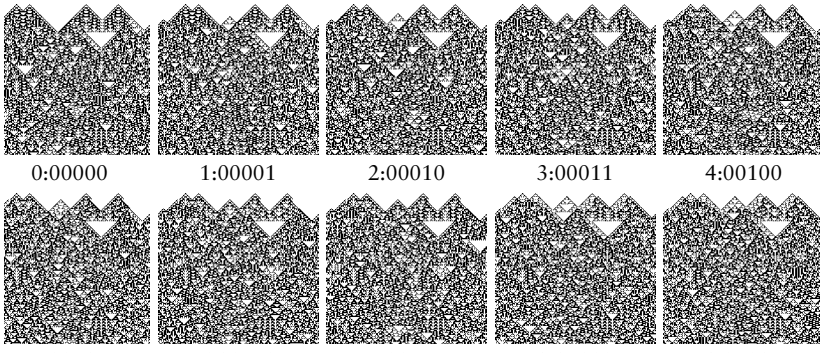

Figure 25. (*continues*)

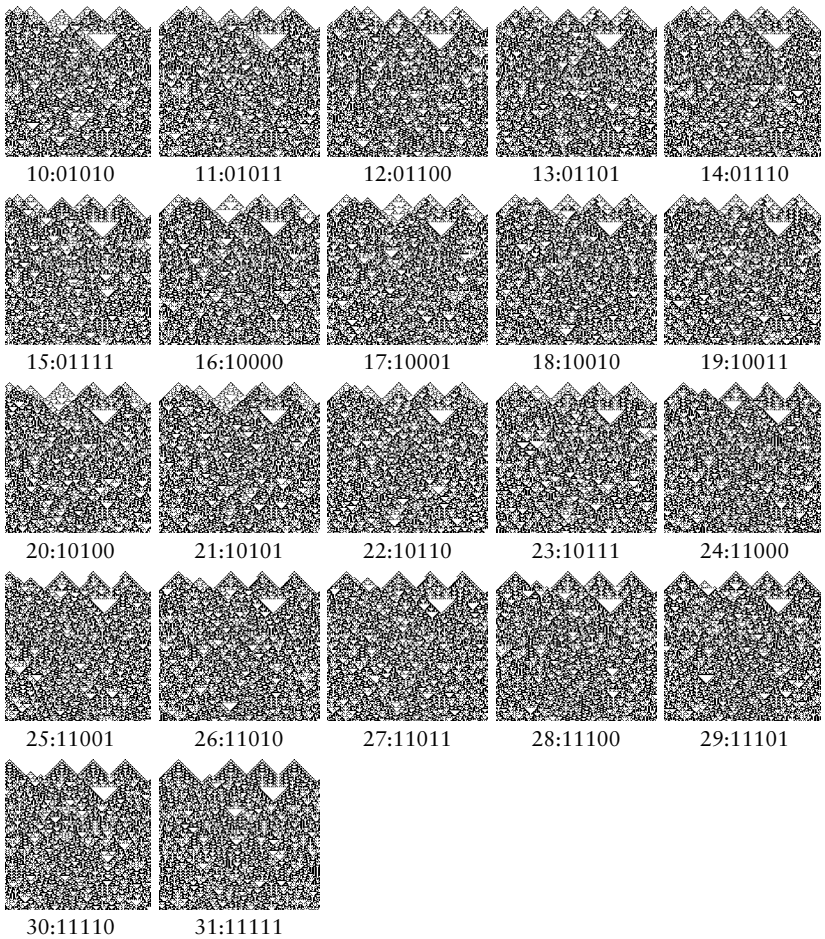

Figure 25. Rule 54, all input, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

5.5.4 Rule 90 and Spacetime Chimera States

Rule 90 is the quintessential elementary rule for ReCAs. In Figure 26, we see an example of its spacetime diagram. At the start of the example, we can see the input creating overlaying triangles that are added together, before turning into what looks like quite chaotic behavior. In Figure 27, we see an interesting behavior where regions are the same in all inputs, this odd version of a strange attractor (regions not perturbed by the different inputs). The figure is converted into a gif and can be found at [48], as well as a version with more steps at [49]. This feature does not occur in every rule 90 run, but usually, we observe larger regions of local spacetime states that are common over

several different inputs but not every single one; such an example can be found at [50]. These regions could potentially give a clue to which step the CA is in, and if it is partially stable, it can help separate the different original input from the current state. This concept is also very reminiscent of chimera states [51–53], a coexistence of coherent (synchronized) and incoherent (desynchronized) first identified in oscillators. In our case, the chimera states are not local to specific oscillators (or CA cells) only, but in space and time.

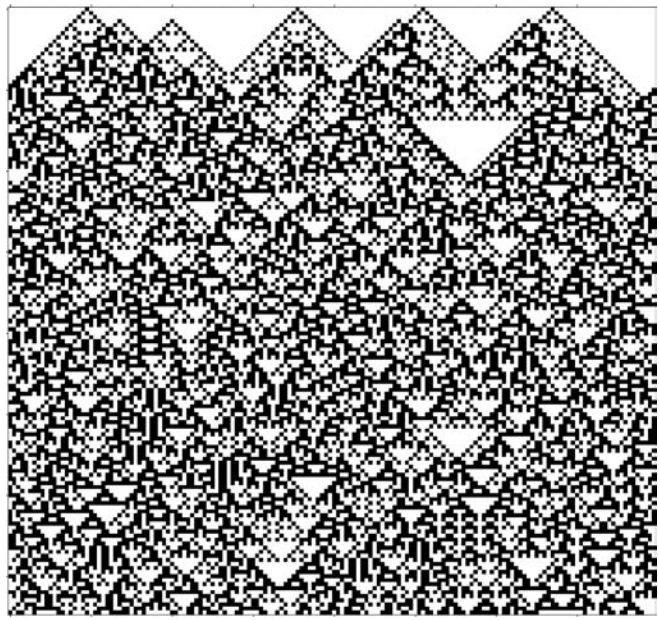

Figure 26. Rule 90, input 10110, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 132.

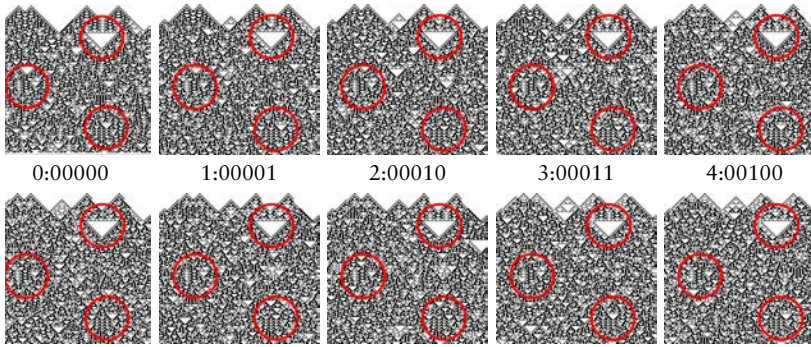

Figure 27. (*continues*)

https://doi.org/10.25088/ComplexSystems.32.3.309

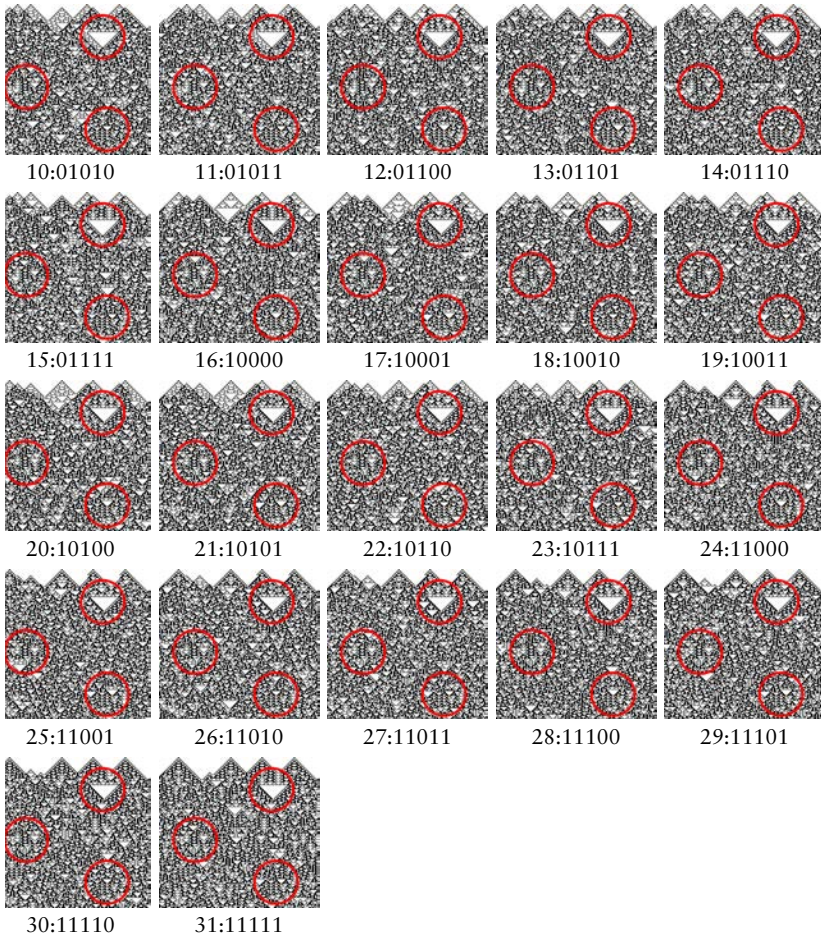

Figure 27. Rule 90, all input, input locations 19, 27, 5, 29, 71, 40, 50, 48, 102, 96, 82, 90, 134, 128, 140, 13; several large stable regions can be observed even at the end of the CA history.

5.5.5 Rule 90 and Initial Condition Do Not Matter in Binary 2^n Grid Sizes

In [12, 20], rule 90's relationship to grid size was pointed out. The randomization period of the rule had a relationship to odd versus even grid sizes, where the longest period was with primary numbers and the shortest was for 2^n for some n. In this section, we see why the binary n^2 grid sizes perform as they do, and what that means for ReCAs. First, let us consider rule 90's differentiation pattern as seen in Figure 3, as we can draw some understanding from it. The differentiation pattern looks just like the rule does if it is initiated with a

single centroid. In combination, we also know due to the Boolean logic of rule 90 that if all cells are black, the next state is all white, or if every other cell is black, the next state is all white as well.

Now, if we consider rule 90 with a central pattern in Figure 1, the example of alternating black and white is happening for the entire perturbed grid. Indeed, at step 3, the central seven cells are alternating, and at step 7, the central 15 cells are alternating and so forth. If we set the CA grid size to match these alternating points such that the grid size is 2^n (2/4/8/16/32/64/...), these alternating cells will cause the entire CA to become quiescent. This remains true not just for centroid initialization, but because of the difference pattern, as seen in Figure 3, the CA for these 2^n grid sizes will always become quiescent within steps. This effect can be seen in Figures 28 and 29. Therefore, for these grid sizes, the memory of any input will never exceed this limit. If we apply this to the x-bit memory benchmark, this means that for this rule and these grid sizes, the CA will always land in an attractor where the original input has no impact on the final CA state, as can be seen in [54]. We can make similar predictions for many of the other additive rules, such as rules 60 and 102 will exhibit the same behavior but after twice as many steps, as they are not bidirectional.

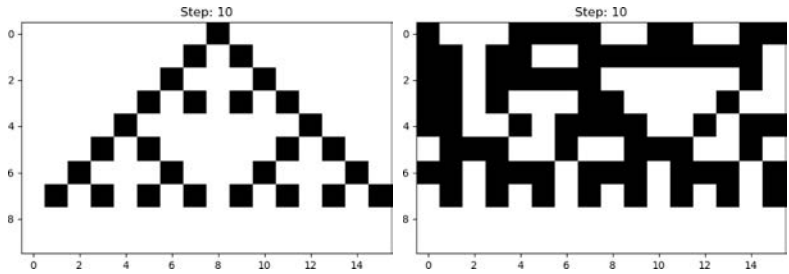

Figure 28. Rule 90 grid size 16, centroid and random initiation.

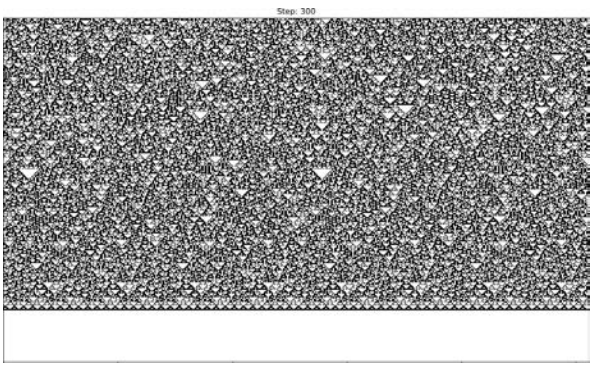

Figure 29. Rule 90 grid size 512, random initiation.

5.5.6 Rule 90 and Why Otherwise Even-Numbered Grid Sizes Perform Well

If we take this understanding of the differentiation pattern of rule 90 and look at grid sizes 39, 40 and 41, as seen in Figure 30, we notice that in one of them the attractor length is very short, yet not in the others. This pattern would mean that this short attractor length would also occur in the reservoir for width 40 in Figure 31. We hypothesize that this is at least one reason why rule 90 (and rules 60 and 150) have such a marked difference in performance on $L_d = 39$ versus $L_d = 40$: the attractor length is short and therefore the SVM has to separate fewer instances.

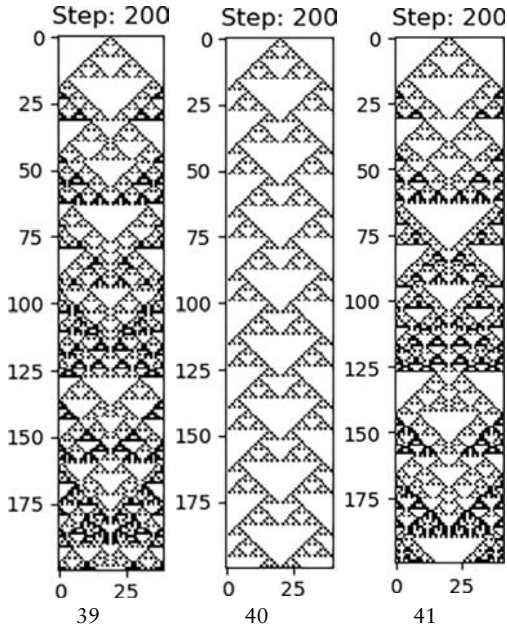

Figure 30. Rule 90 grid size 39 versus 40 versus 41. Grid sizes 39 and 41 feature a longer attractor length, while grid size 40 has a shorter one.

5.5.7 Rule 150

Rule 150, like rule 90, also has this self-similar differentiation pattern, and we find the same answer here for why it performs better at certain grid sizes. As we can see in Figure 32, the attractor in one is much shorter than in the other. We can also make similar predictions to rule 90 on the binary grid size, but due to its pattern, rather than going quiescent, the CA would end up in the same state as its initial condition.

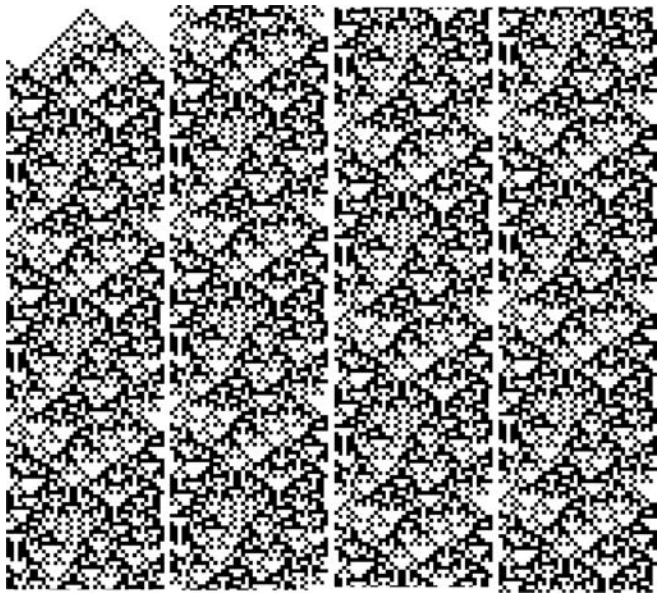

Figure 31. Rule 90 example with $L_d = 40$ in four slices showing a repeating pattern.

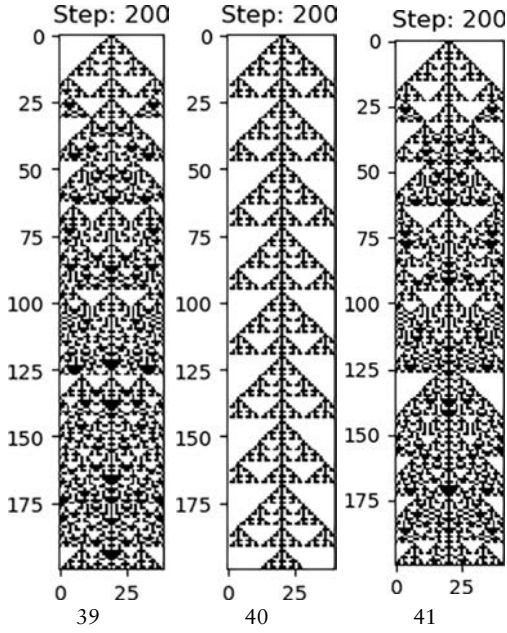

Figure 32. Rule 150 grid sizes 39, 40 and 41. Grid sizes 39 and 41 feature a longer attractor length, while grid size 40 has a shorter one.

https://doi.org/10.25088/ComplexSystems.32.3.309

Note that these rules do not behave quite as chaotically as they were perceived to in [55, p. 227]. Though they are examined under different conditions, this is a much shorter attractor length and puts into question whether these rules deserve to be classified as chaotic under these conditions.

6. Discussion and Conclusion

We have in this and past work conducted the most extensive exploration of ECA parameters for ReCAs and pointed out observations of many results. In this section, we will sum up the results as well as tie them to the greater context of CAs, RC and ReCAs.

6.1 *I* Is about More than Just Reservoir Size

Considering the experiments with varying I in contrast to those with varying D_p and those with simultaneously varying D_p over L_d, some rules, like rules 3, 15 and 56, showed dynamics in the I but not the same trends in the D_p, while none of the rules explored showed dynamical behavior over small increments of D_p in the D_p over L_d experiments. This indicates that the timesteps' length did not much affect the results in these examples, leaving only the conclusion that the highly dynamic behavior in these rules comes from the number of pure CA steps between inputs. So we can say that there are patterns around the parameter of I that are not just about the reservoir size. We can make a similar argument about the other parameters that affect the reservoir size, the R and L_d parameters [15].

6.2 The Importance of Grid Size L_d

We have seen further evidence of how L_d or the grid size will greatly affect performance in some rules, particularly in the rules most studied in literature on ReCAs. We have seen how particular configurations have great importance, such that they can enhance or even break the ability of the reservoir to perform the simple task of 5-bit memory.

6.3 Attractor Length and D_p

We have seen that in certain CAs where their attractor length is already reached, a longer D_p does not hamper performance. Additionally, in many rules, due to the ordered nature of always inputting the distractor at the same locations, the distractor does not strongly affect the reservoir's capabilities. Although exact length sometimes does still affect performance in dynamic ways, we hypothesize this is due to the

exact expansion and contraction of the input such that it hit at better lengths, making either the input or cue signal easier or harder to separate in the reservoir.

6.4 Less Is More, Sometimes

It is natural to assume that, as a general trend, bigger reservoirs mean better performance, but it is important to note that there are other parameters such as the grid size that affect performance. Furthermore, in all parameters that affect grid size or reservoir size (L_d, R and I), we see many exceptions where smaller reservoirs with the same rule outperform the effect of a larger reservoir.

6.5 Patterns within Patterns

We have seen evidence that the performance of rule 90 has patterns that are not simple; there are patterns within patterns of the parameters' performance. That is, checked even numbers seem better, but the worst theoretical performing ones are binary values for rule 90. For this single rule, there are at least two patterns affecting performance. We hypothesize that this is not only true for this single rule and that there are many patterns we have not perceived.

6.6 Not Quite Chaotic

We have also seen how some classified chaotic rules behave noticeably less chaoticically under certain circumstances, at least regarding the attractor length. That means if it were possible to neatly categorize ECAs from chaotic to ordered, this would then depend on the parameters, and different parameters might yield a different order. As was demonstrated for asymptotic behavior for different initial conditions in [26], similarly this can be true for other parameters, for example, the grid size.

Attempts at mapping CA rule space over some variable or function and identifying an edge of chaos point only worked in non-ECA CAs [25]. If mapping ECA rule space over some variable or function and identifying the edge of chaos could be done, then it would need to at least include features such as the grid size and initial condition.

6.7 Pseudorandom Rule 30, Why the Sharp Turn?

As observed in [15] and in Figure 12, the strong chaotic rules exhibit a sharp turn when the 5-bit memory benchmark changes from hard to trivial. Rule 30 is part of this group and is seen as the quintessential pseudorandom number generator rule, particularly its central column [55, p. 315]. This rule and those similar to it would be expected to turn the original five bits, due to the tiny variations between them,

https://doi.org/10.25088/ComplexSystems.32.3.309

into very different configurations of the CA over time (deterministic chaos). Essentially, every configuration that the SVM has to separate is turning pseudorandom and very likely to be unique. Our SVM simply then needs to be able to separate the $5 \times 32 = 160$ (final five output steps of every permutation) final output steps from the remaining $(210 \times 32) - 160 = 6560$ output steps. It appears that if the vector our SVM has access to is large enough ($I \geq 4$), or if the separating classes are easier ($N_b \leq 4$), this turns very possible and quickly trivial. Interestingly, we tried to do the same x-bit memory benchmark, but instead of using a CA, we generated and used a pseudorandom vector of the same size and data type as the CA would have produced; the results can be compared to rule 30 in Table 5. Despite this being a random vector, the SVM could still separate in all instances where rule 30 allowed it to separate. The following quote is attributed to Linus Torvalds: "Given enough eyeballs, all bugs are shallow." If we steal this and apply it to our context, we could say that "given enough dimensions, all classifications are trivial," provided that your substrate is exhibiting sufficient deterministic chaos and that you do not cross-validate or apply similar methods. The conclusion of this might make it seem that we are stating that if you cannot classify, then use more dimensions, but this is more of a warning, if your system is only able to classify because you use so many dimensions that even a pseudorandom number generator can solve it, then will the solution generalize? Moreover, will the solution even scale?

Metric	Rule 30		Random Vector	
	$N_b = 4$	$N_b = 5$	$N_b = 4$	$N_b = 5$
$\overline{w}$	100	7.8	100	9.2
$P\%$	100	0	100	0
	$D_p = 50$	$D_p = 100$	$D_p = 50$	$D_p = 100$
$\overline{w}$	80	49	79	51
$P\%$	0	0	0	0
	$I = 3$	$I = 4$	$I = 3$	$I = 4$
$\overline{w}$	82	100	87	100
$P\%$	0	100	0	100

Table 5. Data for solving the x-bit memory benchmark with a random vector versus rule 30.

6.8 The Limitations of the x-Bit Memory Benchmark

The history of this 5-bit memory benchmark is a bit muddled, but regardless, does this benchmark make sense as the standard of ReCAs? Said differently, does it still select for beneficial properties?

We have identified that the x-bit memory benchmark selects features such as, Can the cue signal be detected?; Do the different inputs lead to different states in the reservoir?; Does the distractor period not perturb the original input of the system? The second one seems a bit analogous to separation property [3] or deterministic chaos, but the third one is conceptually the opposite of the echo state property, as memory will not fade. Are these the right features to ask of a good reservoir? Or rather perhaps the benchmark is not all that useful as a high truth and only is useful as a low bar to pass.

Let us put some of our specific findings in a real example. We just showed how the benchmark can be solved with sufficient dimensions and a random vector. In Table 5, $I = 4$, the vector that the SVM has access to is $I \times R \times L_d = 4 \times 4 \times 40 = 640$ dimensions. In [38], the vector used was $4 \times 8 \times 20 = 640$, meaning at least for $D_p = 200$ the benchmark can be solved using just this feature. This could also further explain why more of the asynchronous ECAs perform better, the introduction of a stochastic order of updates simply pushes the rules to behave more chaotically, allowing them to solve the benchmark using just pseudorandomness.

One alternative is changing the benchmark, say if the distractor period input is placed not in the same location every time, but changing locations, this might be better at demonstrating fading memory. Moreover, the training and testing on the same data mean no cross-validation, but we could simply add cross-validation between different input locations, and it should still be possible to solve in some aspects. Furthermore, maybe the x-bit memory benchmark just does not fit well with CAs, and the concept of fading memory is not a good quality measure for ReCAs, and new measures should be invented in the CA context. Finally, as we have previously hinted at, computation is just computation; its usefulness depends on the context. Therefore, any true usefulness of ReCAs can only be demonstrated by doing something useful with them.

7. Future Work

The dynamical nature of the parameter space presents us with a problem: how do we find good combinations? Considering the many ways that the different parameters can be combined, an exhaustive search is not feasible, but if we can work on good methods to search for a given combination—for example, use an evolutionary algorithm to optimize for good parameters—this could help us utilize the potential of reservoir computing with cellular automata (ReCAs).

We have now looked very deeply at ReCAs in the context of the x-bit memory benchmark, but this benchmark only tests for memory,

and the parameter space would likely look very different for a benchmark that relies more on the complex interaction of the information rather than just its persistence. Therefore, running other experiments using different benchmarks, for example, reinforcement learning in the AI Gym [56], could yield interesting results.

Cellular automata (CAs) are a very specific substrate and can be viewed as a special case of random Boolean networks (RBNs), and we can view RBNs as a special case of neural networks (NNs). In the context of reservoir computing (RC), what is gained or lost by this constrained version? it could be useful to understand CAs and RBNs and even NNs in contrast to each other and RC, to better understand the strength and weaknesses of each substrate.

Acknowledgments

This work was financed mainly by the Research Council of Norway's DeepCA project, grant agreement 286558, and in part by Swedish Research Council project 2022-04657. The work was conducted at the OsloMet Artificial Intelligence Lab and the NordSTAR - Nordic Center for Sustainable and Trustworthy AI Research. Special thanks to Trym Lindell and Barbora Hudcová for useful and inspiring feedback and discussions.

References

[1] E. Strubell, A. Ganesh and A. McCallum, "Energy and Policy Considerations for Deep Learning in NLP." arxiv.org/abs/1906.02243.

[2] H. Jaeger, "The 'Echo State' Approach to Analysing and Training Recurrent Neural Networks: With an Erratum Note," *German National Research Center for Information Technology GMD Technical Report*, report number 148, 2001 p. 13. www.ai.rug.nl/minds/uploads/EchoStatesTechRep.pdf.

[3] W. Maass, T. Natschläger and H. Markram, "Real-Time Computing without Stable States: A New Framework for Neural Computation Based on Perturbations," *Neural Computation*, **14**(11), 2002 pp. 2531–2560. doi:10.1162/089976602760407955.

[4] G. Tanaka, T. Yamane, J. B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano and A. Hirose, "Recent Advances in Physical Reservoir Computing: A Review," *Neural Networks*, **115**, 2019 pp. 100–123. doi:10.1016/j.neunet.2019.03.005.

[5] A. Morán, C. F. Frasser, M. Roca and J. L. Rosselló, "Energy-Efficient Pattern Recognition Hardware with Elementary Cellular Automata," *IEEE Transactions on Computers*, **69**(3), 2019 pp. 392–401. doi:10.1109/TC.2019.2949300.

[6] Y.-L. Lee, P.-K. Tsung and M. Wu, "Technology Trend of Edge AI," in *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, Hsinchu, Taiwan, Piscataway, NJ: IEEE, 2018 pp. 1–2. doi:10.1109/VLSI-DAT.2018.8373244.

[7] A. Moin, A. Zhou, A. Rahimi, A. Menon, S. Benatti, G. Alexandrov, S. Tamakloe, et al. "A Wearable Biosensing System with In-Sensor Adaptive Machine Learning for Hand Gesture Recognition," *Nature Electronics*, **4**(1), 2021 pp. 54–63. doi:10.1038/s41928-020-00510-8.

[8] O. Yilmaz, "Reservoir Computing Using Cellular Automata." arxiv.org/abs/1410.0162.

[9] S. Nichele and A. Molund, "Deep Learning with Cellular Automaton-Based Reservoir Computing," *Complex Systems*, **26**(4), 2017 pp. 319–339. doi:10.25088/ComplexSystems.26.4.319.

[10] S. Nichele and M. S. Gundersen, "Reservoir Computing Using Non-uniform Binary Cellular Automata." arxiv.org/abs/1702.03812.

[11] M. Margem and O. S. Gedik, "Reservoir Computing Based on Cellular Automata (RECA) in Sequence Learning," *Journal of Cellular Automata*, **14**(1–2), 2019 pp. 153–170.

[12] D. Kleyko, E. P. Frady and F. T. Sommer, "Cellular Automata Can Reduce Memory Requirements of Collective-State Computing." arxiv.org/abs/2010.03585.

[13] N. Babson and C. Teuscher, "Reservoir Computing with Complex Cellular Automata," *Complex Systems*, **28**(4), 2019 pp. 433–455. doi:10.25088/ComplexSystems.28.4.433.

[14] N. McDonald, "Reservoir Computing and Extreme Learning Machines Using Pairs of Cellular Automata Rules," in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, Piscataway, NJ: IEEE, 2017 pp. 2429–2436. doi:10.1109/IJCNN.2017.7966151.

[15] T. E. Glover, P. Lind, A. Yazidi, E. Osipov and S. Nichele, "The Dynamical Landscape of Reservoir Computing with Elementary Cellular Automata," in *ALIFE 2021: The 2021 Conference on Artificial Life*, Cambridge, MA: MIT Press, 2021. doi:10.1162/isal_a_00440.

[16] J. von Neumann, *Theory of Self-Reproducing Automata* (A. W. Burks, ed.), Urbana, IL: University of Illinois Press, 1966.

[17] M. Mitchell, "Life and Evolution in Computers," *History and Philosophy of the Life Sciences*, **23**(3/4), 2001 pp. 361–383. www.jstor.org/stable/23332520.

[18] M. Cook, "Universality in Elementary Cellular Automata," *Complex Systems*, **15**(1), 2004 pp. 1–40. complex-systems.com/pdf/15-1-1.pdf.

[19] B. Hudcová and T. Mikolov, "Computational Hierarchy of Elementary Cellular Automata," *ALIFE 2021: The 2021 Conference on Artificial Life*, Cambridge, MA: MIT Press, 2021, p. 105. doi:10.1162/isal_a_00447.

[20] O. Martin, A. M. Odlyzko and S. Wolfram, "Algebraic Properties of Cellular Automata," *Communications in Mathematical Physics*, **93**(2), 1984 pp. 219–258. doi:10.1007/BF01223745.

[21] T. Rowland and E. W. Weisstein, "Additive Cellular Automaton" from Wolfram MathWorld—A Wolfram Web Resource. mathworld.wolfram.com/AdditiveCellularAutomaton.html.

[22] G. J. Martínez, "A Note on Elementary Cellular Automata Classification," *Journal of Cellular Automata*, **8**(3–4), 2013 pp. 233–259.

[23] S. Wolfram, "Universality and Complexity in Cellular Automata," *Physica D: Nonlinear Phenomena*, **10**(1–2), 1984 pp. 1–35. doi:10.1016/0167-2789(84)90245-8.

[24] G. J. Martínez, A. Adamatzky and R. Alonso-Sanz, "Designing Complex Dynamics in Cellular Automata with Memory," *International Journal of Bifurcation and Chaos*, **23**(10), 2013 1330035. doi:10.1142/S0218127413300358.

[25] C. G. Langton, "Computation at the Edge of Chaos: Phase Transitions and Emergent Computation," *Physica D: Nonlinear Phenomena*, **42**(1–3), 1990 pp. 12–37. doi:10.1016/0167-2789(90)90064-V.

[26] H. Zenil and E. Villarreal-Zapata, "Asymptotic Behavior and Ratios of Complexity in Cellular Automata," *International Journal of Bifurcation and Chaos*, **23**(09), 2013 1350159. doi:10.1142/S0218127413501599.

[27] A. N. Kolmogorov, "Three Approaches to the Quantitative Definition of Information," *International Journal of Computer Mathematics*, **2**(1–4), 1968 pp. 157–168. doi:10.1080/00207166808803030.

[28] C. Gallicchio and A. Micheli, "Deep Reservoir Computing: A Critical Analysis," in *24th European Symposium on Artificial Neural Networks (ESANN)*, Bruges (Belgium) (M. Verleysen, ed.), Louvain-la-Neuve, Belgique: Ciaco, 2016.

[29] C. Fernando and S. Sojakka, "Pattern Recognition in a Bucket," in *European Conference on Artificial Life (ECAL 2003)*, Dortmund, Germany (W. Banzhaf, J. Ziegler, T. Christaller, P. Dittrich and J. T. Kim, eds.), Berlin, Heidelberg: Springer, 2003 pp. 588–597. doi:10.1007/978-3-540-39432-7_63.

[30] P. Aaser, M. Knudsen, O. H. Ramstad, R. van de Wijdeven, S. Nichele, I. Sandvig, et al., "Towards Making a Cyborg: A Closed-Loop Reservoir-neuro System," in *Fourteenth European Conference on Artificial Life (ECAL 2017)*, Lyon France, Cambridge, MA: MIT Press, 2017 pp. 430–437.

[31] D. Nikolic, S. Haeusler, W. Singer and W. Maass, "Temporal Dynamics of Information Content Carried by Neurons in the Primary Visual Cortex," in *Advances in Neural Information Processing Systems 19 (NIPS 2006)*, Canada (B. Shölkopf, J. C. Platt and T. Hoffman, eds.), Cambridge, MA: MIT Press, 2006 pp. 1041–1048. papers.nips.cc/paper_files/paper/2006/file/60792d855cd8a912a97711f91a1f155c-Paper.pdf.

[32] X. Dai, "Genetic Regulatory Systems Modeled by Recurrent Neural Network," in *International Symposium on Neural Networks (ISNN 2004)*, Dalian, China (F. L. Yin, J. Wang and C. Guo eds.), Berlin, Heidelberg: Springer, 2004 pp. 519–524. doi:10.1007/978-3-540-28648-6_83.

[33] B. Jones, D. Stekel, J. Rowe and C. Fernando, "Is There a Liquid State Machine in the Bacterium *Escherichia coli*?," in *2007 IEEE Symposium on Artificial Life*, Honolulu, HI, Piscataway, NJ: IEEE, 2007 pp. 187–191. doi:10.1109/ALIFE.2007.367795.

[34] H. Jaeger, "Echo State Network," *Scholarpedia*, **2**(9), 2007 2330. Revision #196567. doi:10.4249/scholarpedia.2330.

[35] L. Grigoryeva and J.-P. Ortega, "Echo State Networks Are Universal," *Neural Networks*, **108**, 2018 pp. 495–508. doi:10.1016/j.neunet.2018.08.025.

[36] W. Olin-Ammentorp, K. Beckmann and N. C. Cady, "Cellular Memristive-Output Reservoir (CMOR)." arxiv.org/abs/1906.06414.

[37] D. Liang, M. Hashimoto and H. Awano, "BloomCA: A Memory Efficient Reservoir Computing Hardware Implementation Using Cellular Automata and Ensemble Bloom Filter," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE 2021)*, Grenoble, France, Piscataway, NJ: IEEE, 2021 pp. 587–590. doi:10.23919/DATE51398.2021.9474047.

[38] D. Uragami and Y.-P. Gunji, "Universal Criticality in Reservoir Computing Using Asynchronous Cellular Automata," *Complex Systems*, **31**(1), 2022 pp. 103–121. doi:10.25088/ComplexSystems.31.1.103.

[39] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, **9**(8), 1997 pp. 1735–1780. doi:10.1162/neco.1997.9.8.1735.

[40] J. Martens and I. Sutskever, "Learning Recurrent Neural Networks with Hessian-Free Optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML '11)*, Bellevue, WA (L. Getoor and T. Scheffer, eds.), Madison, WI: Omnipress, 2011 pp. 1033–1040.

[41] I. Sutskever, *Training Recurrent Neural Networks*, Ph.D. thesis, Department of Computer Science, University of Toronto, Toronto, ON, Canada, 2013.

[42] H. Jaeger, "Long Short-Term Memory in Echo State Networks: Details of a Simulation Study," Technical Report, Jacobs University Bremen, 2012. opus.constructor.university/frontdoor/index/index/docId/638.

[43] S. Pontes-Filho, P. Lind, A. Yazidi, J. Zhang, H. Hammer, G. B. M. Mello, I. Sandvig, G. Tufte and S. Nichele, "A Neuro-inspired General Framework for the Evolution of Stochastic Dynamical Systems: Cellular Automata, Random Boolean Networks and Echo State Networks towards Criticality," *Cognitive Neurodynamics*, **14**(5), 2020 pp. 657–674. doi:10.1007/s11571-020-09600-x.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., "Scikit-learn: Machine Learning in Python," *The Journal of Machine Learning Research*, **12**(85), 2011 pp. 2825–2830.

[45] "ReCA_X-bit_memory." (Nov 28, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/ReCA_X-bit_memory.py.

[46] "Create Graphs." (Nov 28, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/Data/Create%20Graphs.ipynb.

[47] "Trival170Long." (Nov 28, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/examples/Rule170/Trival170Long.gif.

[48] "ChimeraStates." (Dec 21, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/examples/Rule90/ChimeraStates.gif.

[49] "chimeraStates2." (Dec 21, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/examples/Rule90/chimeraStates2.gif.

[50] "RegionsOfDiffTemporalRelations." (Dec 21, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20parameter%20space/examples/Rule90/RegionsOfDiffTemporalRelations.gif.

[51] Y. Kuramoto and D. Battogtokh, "Coexistence of Coherence and Incoherence in Nonlocally Coupled Phase Oscillators." arxiv.org/abs/cond-mat/0210694.

[52] D. M. Abrams and S. H. Strogatz, "Chimera States for Coupled Oscillators," *Physical Review Letters*, **93**(17), 2004 174102. doi:10.1103/PhysRevLett.93.174102.

[53] V. García-Morales, "Cellular Automaton for Chimera States," *Europhysics Letters*, **114**(1), 2016 18002. doi:10.1209/0295-5075/114/18002.

[54] "width32." (Dec 21, 2023) github.com/DeepCANFR/DeepCA---Hybrid-Deep-Learning-Cellular-Automata-Reservoir/blob/master/ReCA%20 parameter%20space/examples/Rule90/width32.gif.

[55] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, Inc., 2002.

[56] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym." arxiv.org/abs/1606.01540.

https://doi.org/10.25088/ComplexSystems.32.3.309