ENERGY DISSIPATION AND ASSOCIATED PROPERTIES OF RESERVOIR COMPUTERS USING RANDOM BOOLEAN NETWORKS

An Honors Thesis

Presented By

William R. Sullivan

Completion Date:
May 2014

Approved By:

_____

Neal Anderson, Electrical and Computer Engineering

_____

Weibo Gong, Electrical and Computer Engineering

ABSTRACT

Title: **Energy Dissipation and Associated Properties of Reservoir Computers Using Random Boolean Networks**
Author: **William R. Sullivan**
Thesis / Project Type: **Independent Honors Thesis**
Approved By: **Neal Anderson, Electrical and Computer Engineering**
Approved By: **Weibo Gong, Electrical and Computer Engineering**

We explore different characteristics of Random Boolean Networks (RBNs) when used as dynamical reservoirs within the reservoir computer (RC) framework. Specifically, we investigate the relationship between the computational capability and a defined lower bound on the average energy dissipated per time step of RC devices with RBN reservoirs (RC-RBNs). We also investigate the mapping between RC-RBNs and finite-state automata (FSA) to show how adjusting parameters in the RC-RBN can change properties in the equivalent FSA. Although other studies have given loose bounds for energy dissipation of classical RBNs, we propose an algorithm which gives a tight lower bound on the average energy dissipated per time step of RC-RBNs. Our algorithm functions by converting RC-RBNs to equivalent FSA, and then applying the measure of energy dissipation to networks which can be converted to irreducible FSA (IFSA). We also present an alteration on this algorithm which gives a tight lower bound on the average energy dissipated per time step of RC-RBNs only after being perturbed by an input string leading to an irreducible-sub FSA (IS-FSA). Due to the large state space of the equivalent FSAs, we focus the majority of our study on RBNs with fewer than 15 nodes (*Small-N* networks). Through this research we have developed two methods of applying this bound onto RC-RBNs, and we present both of them, along with an exploration of the RC-RBN parameter space. This study can be used as a baseline for future studies of energy dissipation in RC-RBN devices.

Contents

1 Introduction

There is an ever increasing demand for faster, smaller and more efficient computers. However, as technology enters the sub-nanometer scale, engineers are forced to rethink the way they develop and manufacture these devices. One significant issue of current fabrication techniques is the unpredictable design variation associated with creating function specific hardware. This restriction has led computer architects in the search for effective, alternative forms of computation. Some of these alternatives include random nano devices and self assembling systems. Construction of these physical systems does not require a traditional top-down engineering approach, but instead, harnesses the intrinsic complexity naturally available in certain nano structures. To better understand how these systems can be used in computing, researchers have used the *Reservoir Computing (RC)* model as an analysis tool. Although there has been much research in terms of RC as a whole, there is very little research in terms of RC when applied as a model to these types of physical systems.

The RC paradigm is a neural-network computational framework that uses a fixed, dynamical system called a *reservoir* to perform computation. The first developed RC methods were *Echo State Networks (ESNs)* and *Liquid State Machines (LSMs)* which were created simultaneously, yet, independently of each other [1]. This model was first introduced at the start of the 21st century, and its development came as a welcome alternative to classical methods of training *Recurrent Neural Networks RNNs*. Prior to RC, RNNs were trained using methods well suited for *Feedforward Neural Networks (FFNNs)*, but the cyclic dependencies in RNNs often rendered RNNs untrainable due to their unpredictable feedback. RC removes these cyclic dependencies by training only a readout layer connected to the RNN, and thus removing the necessity to update weights internal to the RNN [2]. Although most research on RC is focused on RNNs, there have been studies which suggest that the RC paradigm offers a promising alternative form of computation when used to model physical computing devices [3].

While RC devices benefit from both simplified training methods and in their general

reservoir requirements, there has not yet been thorough research in the use of reservoirs which are well suited for random nano systems. In their 2013 paper, Snyder, Goudarzi and Teuscher (hereafter SGT) mention that research should be conducted in modeling the *physical implementations* of RC devices, because self assembled systems are subject to uncontrollable design variation [4]. This research gap is the primary motivation for our research and also the primary motivation for the research of SGT. With more thorough research, the generality of these networks may allow engineers to apply RC techniques to a large variety of physical systems which exhibit behavior which may not be preferable in terms of traditional, top-down engineering methodology, but may offer promise in unconventional computing, such as random nano systems.

In our research, we focus on simulating RC devices which employ the use of *Random Boolean Networks (RBNs)* as their internal reservoir. The motivation behind this choice lies in the fact that RBNs can be modeled with heterogenous connectivity while also acting as discrete and dynamic reservoirs with Boolean activation functions. We look to extend the work of SGT, which analyzed the *perceived* computational capability (ability to separate two input strings and eventually forget past perturbations, henceforth *computational capability*), general properties and ability to perform actual computation of *RC devices with RBN reservoirs (RC-RBNs)*. In our research, we only focus on computational capability and disregard any actual outputs or training methodology. We compare computational capability with a measure of energy dissipation in RC-RBNs. To perform this analysis, we first convert the networks into equivalent *Finite-State Automata (FSA)*, where FSA is defined formally in Sec. 1.1.3. Once a network has been represented as an *equivalent FSA*, we apply an information-theoretic minimum bound on the average energy dissipated per time step. This bound is loosely based on Landauer's principle which asserts the necessity for a minimum physical cost associated with the erasure of a logical bit of information from a system. This limit is often called the *Landauer Limit*, and it states that the erasure (or destruction through combination) of n logical bits will cost at least $kTln(2)$ Joules of energy where k is the

Boltzmann constant, T is the temperature of the room in Kelvin and $ln(2)$ is the natural logarithm of 2 [5]. In our study, we focus on the following three questions:

1. How can we obtain a minimum bound on the average energy dissipated per time step for any instance of an RC-RBN, and how is this related to computational capability?

2. How do the parameters of an RC-RBN affect its equivalent FSA, and how do properties of the FSA relate to the dynamics of the original RC-RBN?

3. How does the phase of an RBN (ordered, critical, chaotic) affect the minimum bound on the average energy dissipated per time step for that RC-RBN?

The remainder of this thesis is organized as follows: The remainder of this section works to formalize the abstract concepts of RBNs, RC and FSA, it and also introduces terminology and language used throughout this work. Sec. 1.3 gives a complete review of our research goals. Here we take the general questions listed above and break them down into discrete, quantifiable goals. In Sec. 2, we describe our research methodology. This includes the measures, tools, reasoning and issues we had in accomplishing our objectives. Sec. 3 presents the results of our study as a compilation of four distinct investigations to the RC-RBN model. Sec. 4 brings the thesis to conclusion by summarizing our work, our motivation and giving some perspective to our findings.

1.1 Technical Background

1.1.1 Random Boolean Neworks

RBNs are generalizations of *cellular automata (CA)* which were initially discovered by Stanislaw Ulam and John von Neumann in the 1940s. An RBN can be considered a generalization of a CA because the state of each node is not dependent on only its neighbors, but can be dependent on any other node in the network (including itself)[6]. During the 1950s and the 1960s there was not much research done with concept of CA, but in 1969 Stuart Kauffman

[7] first introduced the idea of RBNs. Here he explains what is now known as the *Kauffman* or, *Classical* RBN model. These networks were initially created to model genetic regulatory networks, but their usage in other applications soon emerged as their general structure can be used to analyze a wide variety of complex, non-linear physical phenomena.

By definition, an instance of a classical (Kauffman) RBN is an interconnected network of Boolean nodes where each instantiation of an RBN has N boolean nodes: $x_i \in \{0, 1\}, i = 0, ..., N-1$. Each of these nodes update in unison with all other nodes in the network and the updating occurs synchronously. Each node has its own Boolean function, f_i, which determines its output at time *t+1*. This function is dependent on the outputs of K_i nodes in the network at time t, $f_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), ..., x_{i_{k_i}}(t))$. The function can be represented as a truth table, where the rightmost column corresponds to the function's output and the leftmost columns correspond to the function's inputs [7]. To better understand this model, Fig. 1 gives an example of a network with $N = 4$ nodes and $K = 2$ input connections per node. The randomly chosen logic functions for this network are shown in Tbl. 1 - 4.

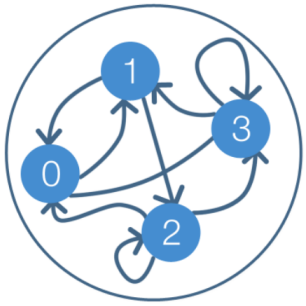

Figure 1: An example of a classic RBN (selected randomly for illustrative purposes) with $N = 4$ nodes and and $K = 2$ inputs to each node in the network. Generally, the internal links of the network are generated randomly using only the given parameters. Because the parameter N was given a value of "4", there are 4 nodes in this network. Because parameter K was given a value of "2", each node has an *in-degree* of two. Each of these nodes has an associated logic function which depends on the arrows pointing *into* the node. These can be seen in Tbl. 1 - 4.

Table 1

1	2	O
0	0	0
0	1	1
1	0	1
1	1	1

Node 0 Function

Table 2

0	3	O
0	0	0
0	1	1
1	0	0
1	1	1

Node 1 Function

Table 3

1	2	O
0	0	0
0	1	1
1	0	0
1	1	0

Node 2 Function

Table 4

2	3	O
0	0	1
0	1	1
1	0	0
1	1	0

Node 3 Function

When generating an RBN, the parameters N and K are set prior to its generation and are chosen decisively. In some models, the parameter p is also set prior to the network's generation and chosen decisively. These parameters are defined such that: N is the number of nodes in the network, K is the in-degree of each node (the number of inputs to each node's Boolean function) and p is the the probability that each node's Boolean function will output a "1" and it is called the *bias* of the function info, power. When used as the parameter in the setup of an RBN, p gives a macroscopic bias which adjusts the distribution used to "select" each node's Boolean function. When p is not used as a parameter to the setup of an RBN, its value defaults to 0.5 such that the distribution used to "select" each node's Boolean function will be uniform.

Although the in-degree, K, is constant for all nodes in the classical Kauffman, it is also possible to create a non-homogeneous variation with which the *average* in-degree of each node is specified, instead of the *absolute* in-degree. In this case the symbol $<K>$ will represent the average in-degree [8]. After the parameters are set up, the links of the RBN need to be randomly generated and the Boolean functions need to be randomly chosen. If a value of K is specified, then an iteration is performed on the set of nodes. Each node will randomly select K nodes from all N nodes in the network from some distribution (usually uniform). These selected nodes will be used as the inputs to the Boolean function of the selecting node. If a value of $<K>$ is used then non-integer values are allowed. In this case, an iteration is performed on the set of nodes as if $K = K_{floor} = floor(<K>)$. Then, the iteration is

performed for an additional $K_{decimal}$ links where $K_{decimal} = floor((<K> - K_{floor}) * N)$.

Once the links are generated, the Boolean functions need to be selected. Since there are K_i Boolean inputs to node x_i, the function f_i has 2^{K_i} possible input configurations. Each of these input configurations has two possible outputs, and therefore there are a total of $2^{2^{K_i}}$ functions for each node, x_i. To assign a function to each node, one of these functions is chosen randomly from some distribution (uniform if $p = 0.5$). After the function is chosen, an RBN is initialized into a given *state*, where its state is defined in terms of the outputs of all its Boolean nodes. We denote the state of the network at time step t as: $S(t)$. For example, if the RBN in Fig. 1 were given an initial state of 0110 and we define the least significant bit of this state as the output of the node with the smallest node index, i, then $\{x_3, x_2, x_1, x_0\} = \{0, 1, 1, 0\}$.

1.1.2 Reservoir Computing with Random Boolean Networks

Reservoir Computing (RC) is a computational framework which uses an input layer, a dynamical network of nodes and a output layer to perform computation. A reservoir is a fixed, excitable network with dynamics generated at the networks creation. These dynamics remain static for the reservoir's life and they dictate the behavior of the network. The output comes from a readout layer which is trained to perform a certain task. In [4], SGT showed that RBNs could operate as reservoirs due to their generality, and in our research, we also use RBNs as reservoirs to the RC model.

To function in an RC environment, an instance of an RBN is connected to both an input layer and an output layer. The input layer perturbs the RBN with a binary stream, and the RBN connects to an output layer which is trained with a regression function. In our research, the input is usually generated by an *independent, identically distributed random variable (IID)* source such that each realization of an input symbol is "1" with probability 0.5 and "0" with probability 0.5.

In contrast with the classical RBN model presented in Sec. 1.1.1, an input layer links to

a specific number of nodes, L, such that nodes which connect to the input layer are given a new update function: $f_i(t+1) = f_i(x_{i_1}(t), x_{i_2}(t), ..., x_{i_{k_i}}(t), \mathcal{I}(t))$ where $\mathcal{I}(t)$ is the value of the input string at time t. It is important to note that these input connections do not affect the values of K or $<K>$, because the input connections are not fully internal to the RBN. The output node is typically defined in the classic machine learning sense such that the output of this node, $\mathcal{O}$, at time t is denoted y_o^t and is computed as

$$y_o^t = sgn(\sum_{j=1}^{N} \alpha_j x_j^t + b) \tag{1}$$

where α_j is a "weight" for the input from node j to the output and b is a bias term which offsets all the weighted inputs by some constant value. Unlike RNNs, these parameters (which are usually included throughout the entire network) are trained directly at the output node using a *readout layer* [4]. Although the question of how to efficiently train RC-RBNs remains open, in our research, we disregard this output node completely and focus only on the internal reservoirs and input node. An example of an RC device using the classical RBN from Fig. 1 can be seen in Fig. 2. This extension to the classical RBN was first studied by SGT, and it is the model we analyze in our research.

1.1.3 RC-RBN and Finite State Automata

An RC-RBN can be fully described as a *finite-state automata (FSA)*, and because of this, it is possible to determine the physical costs associated with the actions of any given RC-RBN [10]. To analyze an instance of an RC-RBN in terms of these physical costs, it needs to first be converted into an equivalent FSA where an FSA is formally defined as as

$$\mathcal{F} \equiv \{\{\sigma\}, \{x\}, \{\mathcal{L}\}\} \tag{2}$$

where σ is a finite set containing all the FSA states, x is a set of input symbols, and $\mathcal{L}$ is a set of state transition rules. Given some input, x_j there exists a transition rule, $\mathcal{L}$ which

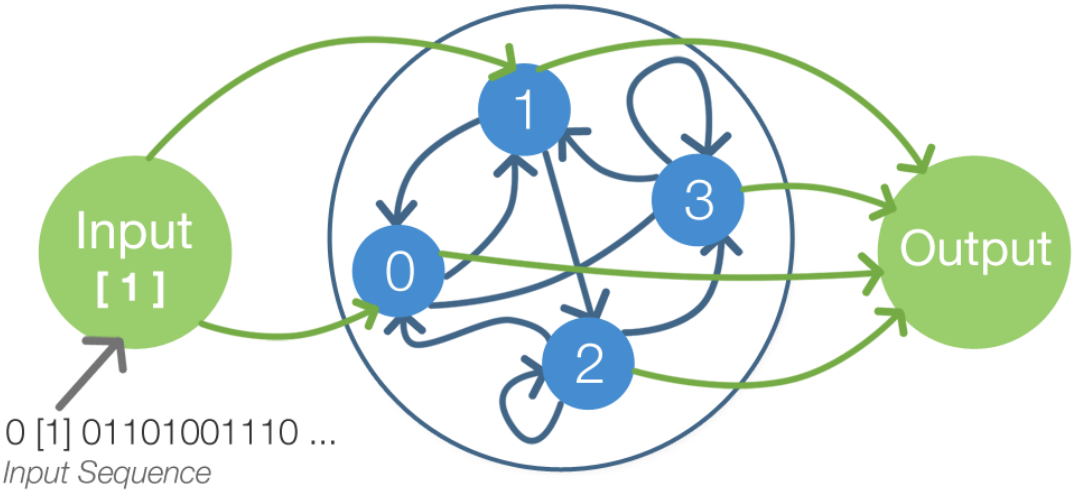

Figure 2: Example of an RC-RBN device using the classical RBN from Fig. 1 as its dynamic reservoir. This figure shows the input layer on the left which is sequentially updated with the next value in a binary input string (usually an IID source). Values on each input line are updated sequentially at each time-step of the RC-RBN device. The number of inputs coming from the input layer is set by the parameter, L ($L = 2$ in this example), and all nodes in the network connect to the output layer.

maps the current state, σ_k to a next state, $\sigma'_{kj} = \mathcal{L}_j(\sigma_k) \in \sigma$ [10]. The FSA generalizes the behavior of the RBN, and at this level, the FSA conveniently captures all irreversible (and thus dissipative) operations performed by the RBN. Although outputs are also usually defined we disregard them as our work focuses only on the transitions between states.

Before generating an RC-RBN, the parameter N needs to be set. Because of this, the generated RC-RBN will have N nodes, and the equivalent FSA will have a total of 2^N states. As the RC-RBN updates through time, the state changes, and so does the output. When an RBN is used as the reservoir to an RC device, the entire system can be modeled as a Moore finite-state machine because the input layer selects the next state, and the output depends only on the current state. We are able to convert an RC-RBN to an FSA with Alg.

2 described in the Appendix.

There are a few terms relating to FSA the reader should be familiar with. These are the terms *irreducible FSA (IFSA)* and *irreducible-sub FSA (IS-FSA)*. IFSA are simply defined as a subclass of all possible FSA, where every state in the FSA is in the same *communicating class*. This means every state in the FSA is capable of reaching every other state in the FSA after some finite number of steps. Of course, most FSA do not belong to this subclass, and this is where the term IS-FSA comes into play. IS-FSA are groups of states within an FSA which, if removed, will create an independent IFSA. This means that within a set of IS-FSA states, any state can be reached from any other state within that set after a finite number of time steps, and can only reach states within that set. It follows that once an IS-FSA state is reached, the only reachable states after that point are also states which are part of that IS-FSA.

1.2 Previous Work

In this section we present a summary of the history, concepts, language and methodology of previous RBN and RC-RBN research. While we primarily focus on properties of classical RBNs and their classification, we also give an overview of simulation methodology in classical RBNs, energy dissipation in classical RBNs and computational capability of RC-RBNs. It should be noted that in 2004, Gershenson [8] presents a thorough review of classical RBNs, and it is cited often throughout this review.

1.2.1 History of Classical Random Boolean Networks

The original paper first defining "RBN" was published by [7] in 1969. This first model has become known as the *Kauffman* or *Classical* RBN model. Here, Kauffman does not use RBNs as a dynamic reservoir in an RC device, but only uses the networks as a generalization of CA which are not perturbed by input strings. Yet, he does allude to the idea of damage and perturbation spreading in these networks, not by an input layer, but by directly altering

the value a node in the network as if caused by some noise in the system. The classical RBN model was already presented in Sec. 1.1.1, but to summarize, an RBN modeled with the classical (Kauffman) model has the following specifications:

- N nodes which generate either a zero or a one and put that symbol on their output connections.
- An in-degree K for each Boolean node in the network.
- Each node's connections are chosen with uniform probability from the nodes (including itself) in the network. These connections remain fixed for the network's lifespan.
- Each node has a randomly generated logic function which acts as a lookup table for the node's inputs. This table remains fixed for the network's lifespan.
- Node updates occur synchronously (discrete time steps).
- The initial state of the network is chosen at initialization.

Because a network following the classical RBN model is not perturbed by an input sequence and the network has a finite number of states, eventually the network's state will have to repeat itself. This causes a chain of states which will be continuously repeated. Such a chain is called an *attractor* [8]. If an attractor contains only one state, then it is called a *point attractor*. If an attractor contains more than one state it is called a *cycle attractor* or *state cycle*. The notion of attractors and their implications are an important area of research in RBNs, as such, there are important definitions for properties of attractors: A series of states which lead to an attractor is called the *trajectory* of the attractor, and many trajectories may lead to the same attractor. A set of states which includes an attractor, $\mathcal{A}$, and all trajectories leading to $\mathcal{A}$ is called the *basin of attraction* of $\mathcal{A}$. The set of all attraction basins is called the *attraction field* [11].

Kauffman continued working with the classical model of RBNs, and in 2000 he published some important classification findings [12]. He explained that RBNs can be classified by

their known, fixed parameters, and that these parameters are indicative of the network's behavior. His research led to three general RBN phase classifications: ordered, critical and chaotic. The phase of an RBN can be determined either graphically or using an analytic formula. For example, after running a network with unknown parameters, one may find the network quickly stabilizes with no patterns (same output every time step), has a set of recurring patterns of some fixed period, or takes a long time to stabilize with noisy patterns. These behaviors represent the ordered, critical and chaotic phases respectively.

In [8], two terms are used which aid in the visualization of these phenomena. These terms are *island* and *frozen sea*. Islands are localized groups of changing nodes while the unchanging nodes surrounding these islands are joined together in a frozen sea. To visualize this behavior, Fig. 3 shows the *change space* of an RBN. The x-axis has N points which correspond to the N nodes in the reservoir and the y-axis has $\triangle t$ points, where $\triangle t$ is the number of time steps from $t = 0$. Each plot point will be blue or white. A blue plot point represents a node which has changed from its previous state, and a white plot point represents a node which has not changed its state. When a network's plot changes from a predominantly white sea with blue islands (ordered), to a blue sea, with small white islands (chaotic), the network transitions through the *edge of chaos* and goes through a critical phase. Fig. 3 shows an example using an RC-RBN with $N = 90$ showing the change map for different values of $<K>$ running on an IID source.

Although it is possible to determine the phase of an RBN graphically, a visual solution is not as robust as a strict analytical classification. So, In 1997 Luque and Sole [13] presented an analytic method in the determination of RBN phase transitions by studying the way perturbations in different phases affect other nodes in the network. Their results were verified, and the following relationships allow RBNs to be classified

$$\lambda = log[2p(1-p)K] \tag{3}$$

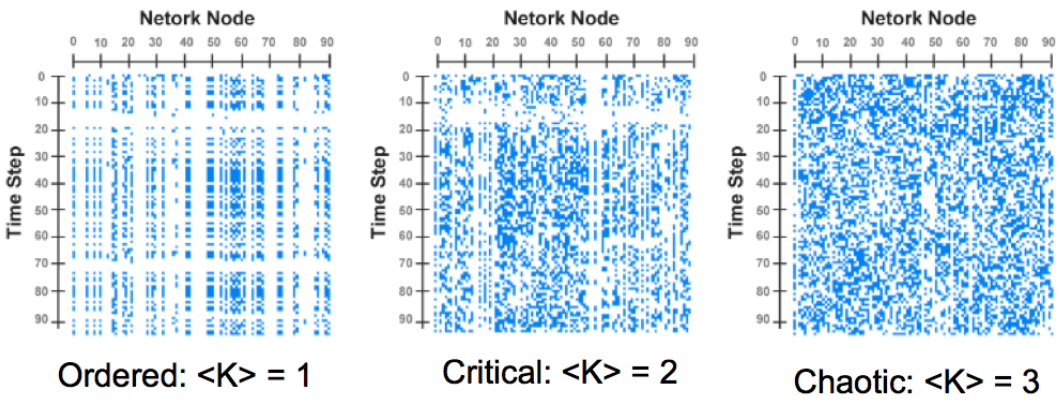

Figure 3: Example of an RBN *change map* which shows nodes which have changed since the previous step (blue) and nodes which have not changed from the previous state (white). The above figure uses $N = 90$ and shows different values of $<K>$ for an IID source. At time zero all the charts show the same identical line of horizontal pixels corresponding to state *zero*. As time moves forward, the values are updated and the states are shown projecting downward.

- Ordered Phase: $\lambda < 0$
- Critical Phase: $\lambda = 0$
- Chaotic Phase: $\lambda > 0$

where λ is representative of the phase of the RBN. Gershenson points out an interesting argument made by researchers regarding the above observations: It seems that for a model network to simulate life (and possibly computation), there needs to be some degree of freedom yet some stability. Therefore networks at criticality are often assumed to provide the best mix of stability and instability [8].

In 1986, Derrida and Pomeau [14] analytically showed that when the connectivity between internal reservoir nodes in a classical network is $K = 2$, the network will be in the critical phase when $p = 0.5$ (where p is the network bias). They then went on to show that this is true for heterogeneous networks as well (networks with average in-degree, $<K>$). These classifications of phase are important assets to any study of RBNs.

While some researchers were interested in classifying the phases of RBNs, others were interested in other properties. In their 1997 paper, Harbery and Bossomaier [15] showed that the number of classical RBNs which can be constructed with parameters N and K can be expressed as

$$\left(\frac{2^{2^K} N!}{(N-K)!} \right)^N \tag{4}$$

which is a huge space for classical RBNs alone. When an input layer is added and the network is adopted to allow non-homogenous in-degrees, this space becomes even larger. Because of this, most researchers, including ourselves, have chosen to approach the study of RBNs through *representative statistics*. In other words, we cannot hope to analyze the full network space, so instead, we use a simulator to perform studies on only a subset of all the networks. We direct our studies to experiments which appear to be representative of the larger space [8].

1.2.2 Simulation Methodology

Researchers have been developing their own custom RBN simulation suites, and some of these programs are listed at the end of [8]. However, in 2007 Harwick [16] published methods for simulating not just RBNs in general, but on simulating large RBNs. The paper provides an overview of high performance code and data structures for simulating large RBNs. The writers use the D programming language and step through the different phases of programming an RBN. Since an RBN is essentially a huge array of changing bits, some attention is given to choosing of a proper representational type for the bit. The paper also mentions that updates should be synchronous and perform a simple switch from a current state to a new state. The paper serves as a collection of useful algorithms with which to use in studying RBNs via simulation.

The reason large RBNs may be interesting in our study is because certain properties appear as the size of an RBN increases [8]. This means that network size may play a role

in the energy efficiency of RC-RBNs. If a large network can perform a computation more efficiently than a medium or small network, then this ability may be noticed through simulation. Unfortunately, as was previously mentioned, our conversion to FSA makes studying large RBNs time consuming, thus, in our study we focus on smaller networks.

1.2.3 Energy Dissipation

In their 2008 paper, Carteret, Rose and Kauffman [9] (henceforth CRK) developed an analytical, mean-field expression for the *Power Efficiency* of RBNs using Landauer's erasure principle. The primary solution of the CRK study conjectures a direct link between RBN criticality, maximum power efficiency and maximum pairwise mutual information. We henceforth refer to this conjecture as the *"Carteret-Rose-Kauffman (CRK) Conjecture"*.

In contrast with our work, the CRK study treats the nodes in the RBN as finite *Turing Machine (TM)* nodes instead of converting the networks to equivalent FSA. In the TM representation, the inputs to each node, (K_i), also represent the inputs to a TM which computes the node's Boolean function in finite time. The CRK study uses this representation and applies previous work done by Bennett [17]. His work made it possible to analyze these TM nodes at finite speeds which gives a more realistic interpretation to the measure of power dissipation. Specifically, the CRK uses the *Gouy-Stodola theorem* [18] to measure the power efficiency of the RBNs as a more "natural" measure. In terms of our research though, the most interesting part of the CRK study was the methodology used in calculating the expected rate of information loss per node.

In the letter, the calculation for expected rate of information loss per node is based on a few different RBN quantities which have not only been influential in the CRK letter, but also in the field of RBN research as a whole. These measures include the previously defined (Sec. 1.1.1) *Node Bias (p)*, and also *Activity (α)* and *Sensitivity (s)* which were both introduced by Shmulevish and Kauffman [19] (henceforth SK) in 2004. Both measures, s and α, are based on the fact that some variables to each node's Boolean function have a

greater influence on the output than other variables. To formalize this point, the SK results give an expression to determine how a Boolean function changes with respect to a certain input. Intuitively, this expression can be interpreted as the partial derivative of the Boolean function, $f : \{0,1\}^K \rightarrow \{0,1\}$ of node x_i with respect to some input, $x_{i_j} = d_j$ such that

$$\frac{\delta f(\mathbf{d})}{\delta d_j} = f(\mathbf{d}^{(j,0)}) \oplus f(\mathbf{d}^{j,1}) \tag{5}$$

where the in-degrees of node x_i form the vector $\mathbf{d} = \{x_{i_1}, x_{i_2}, \ldots, x_{i_{(k_i=K)}}\}$, $\oplus$ is the modulo 2 (XOR) addition, and the vector $\mathbf{d}^{j,k} = (d_1, \ldots, d_{j-1}, k, d_{j+1}, \ldots, d_K), k \in \{0,1\})$, K $\equiv$ the homogenous in-degree of each network node, x_i. With the partial derivative defined, activity can be informally described as the probability an input, d_j to node x_i will be capable of changing the value the node's Boolean function alone. More formally, activity is defined as

$$\alpha_j^f = \frac{1}{2^K} \sum_{\mathbf{d} \in 0,1^K} \frac{\delta f(\mathbf{d})}{\delta d_j} \tag{6}$$

where α_j^f is the activity of variable d_j which has function f. This measure has been used in genetic regulatory network modeling [19, 20].

Now, an intuitive way to define sensitivity, is

$$S^f = \sum_{i=1}^{K} \alpha_i^f \tag{7}$$

but, only for inputs which are randomly selected from a uniform distribution. In this way, sensitivity gives the expected number of inputs of the K Boolean inputs to function f which are capable of changing the value of f alone. To extend the use of these measures further, the

SK results explain a very important yet simple concept: If an RBN has a bias of p and each node has K inputs, then the associated Boolean function will have 2^K entries in its truth table. One often asked question is, "What is the probability that two *Hamming Neighbors* in the input vector will correspond to different output values?" Where two binary strings are *Hamming Neighbors* if, and only if the strings differ by only one bit. To answer this question, one realizes that given two Hamming neighbors, the probability that one will be "1" and the other will be "0" is $2(p)(1-p)$ [19]. The implication of this simple fact is the reasoning behind why the expected value of all the activities of a Boolean function will be $2(p)(1-p)$ regardless of distribution. This also means that the expected sensitivity can be defined as

$$E[S^f] = \sum_{i=1}^{K} E[\alpha_i^f] = \sum_{i=1}^{K} 2(p)(1-p) = K2(p)(1-p). \tag{8}$$

This is a widely quoted figure and important in the way the CRK methodology defines power efficiency. Specifically, the CRK work uses these quantities of sensitivity, activity and RBN bias to show that the expected rate of information loss per node is

$$A_{out} = k(1-\alpha) + 1 = k - k\alpha + 1 = 1 + k - s. \tag{9}$$

where k is the average in-degree of all nodes, $\alpha = E[\alpha_i^f]$, and $s = E[s^f]$. A_{out} is used in the definition of *Specific Dissipation Rate per Node* which is $\mathcal{D} := |A_{in} - A_{out}/A_{in}|$ where A_{in} is the expected number of edges which will be prepared per node, and A_{out} is the expected number of bits of information which are irreversibly erased from each node [9].

One note though, the CRK letter often refers to *power efficiency*, and in the context of the CRK work, the Gouy-Stodola theorem allowed for the expression of

$$E_p = 1 - \frac{T\triangle S}{\triangle W_{rev}} \tag{10}$$

where $T\triangle S$ is the energy (Joules per time step) lost to the environment from irreversible erasure, and $\triangle W_{rev}$ is the work available as input (Joules per time step). This gives a dimensionless measure CRK calls *Power Efficiency (E_p)*. As "power", strictly speaking, is defined as energy (Joules) dissipated per *second*, the CRK definition of power efficiency should be differentiated from the classical definition. Because of this, we refer to *power efficiency* as *Energy dissipated per time step*, but for all intents and purposes, the two are equivalent in the scope of our paper.

Although the specific methodology used in the CRK study differs form our own, the underlying principles (Landauer's principle, questions of criticality, erasure rates, computation speed, etc...) remain the same. The concluding CRK conjecture will lead us in our own questions about critically and energy dissipation. While the CRK study approaches the question of energy dissipation on a per-node bases of the wiring diagram for classical RBNs, we focus on energy dissipation in terms of the state state diagram (FSA) for RC-RBNs.

In their 2013 paper, Ganesh and Anderson [10] (hereafter GA) present a "quantitative measure for the computational irreversibility of finite automata" and a"fundamental lower bound on the average energy dissipated per state transition". The primary result presented in this work is a theorem which states that the input-averaged amount of energy dissipated on each transition of a physical IFSA is lower bounded by

$$\triangle <E^B> \geq k_B T ln(2) \sum_j q_j (I^{R_0 S} - I_j^{R_0 S'}) \tag{11}$$

where B is a the thermal environment, k_B is the Boltzmann constant, T is the temperature of the environment in Kelvin, $I^{R_0 S} - I_j^{R_0 S'}$ is the reduction in mutual information between the state of a physical register system S and the sequence of all past inputs physically

instantiated in a physical referent system R and q_j is the input pmf for a physical IFSA [10]. This bound is not defined for general FSA. Therefore, we limit our study to IFSA mapping RC-RBNs and RC-RBNs which lead to IS-FSA states after some fixed length input. After presenting this minimum bound on the average energy dissipated per state transition, GA proceeds by calling attention to the fact that the bound is proportional to

$$-<\triangle\mathcal{I}_j^{\mathcal{R}_0\mathcal{S}}> = \sum_j q_j H_j(S^{(n-1)}|S^{(n)}) \tag{12}$$

where $-<\triangle\mathcal{I}_j^{\mathcal{R}_0\mathcal{S}}>$ is the "input-averaged amount of physical information about $\mathcal{R}_0$ that is lost from $\mathcal{S}$ on each state transition" [10]. q is defined as the input pmf for the IFSA, $H_j(X|Y)$ is the conditional Shannon entropy of the two random variables X and Y, $S^{(n-1)}$ is the state of the FSA at the $(n-1)$ time step and S^n is defined as the state of the FSA at the n^{th} time step. Therefore, from the definition of conditional Shannon entropy we have

$$H_j(S^{(n-1)}|S^{(n)}) = -\sum_{k\in S} p(y) \sum_{k'\in S_j^k} p(k'|k) log(p(k'|k)) \tag{13}$$

where S is defined as the set of all states, and $S_j^k \subset S$ such that S_j^k represents all states which could have preceded state k. This definition can then be expanded as follows,

$$-<\triangle\mathcal{I}_j^{\mathcal{R}_0\mathcal{S}}> = -\sum_j q_j \sum_{k\in S} p(y) \sum_{k'\in S_j^k} p(k'|k) log(p(k'|k)) \tag{14}$$

This summation is performed on a given IFSA to compute what can be heuristically interpreted as "the average amount of information that the machine state loses about its own history on each transition" [10]. With this interpretation, and fitting with its function in the presented theorem, the unit to this quantity should be the *bit*. With this interpretation, $k_B T ln(2)$ must have units of *Joules / Bit*. Our measure of energy dissipation in this work is equivalent to this value, represented in units of $k_B T ln(2)$. Therefore we require an algorithm to compute this summation, and such an algorithm can be found in the Appendix under

Alg. 1.

1.2.4 Measures for RC-RBNs

In [4], SGT investigates discrete and heterogeneous RBNs at the edge of chaos extended with an input layer meant to avoid attractors (refer to Sec. 1.1.2) [4]. As has been mentioned, the motivation for our work follows closely with that of the SGT work. To reiterate, there has not been much research in the study of RC devices using reservoirs with *heterogenous* in-degrees, as most work is concerned with the training of RNNs (normally homogenous) reservoirs. The study of heterogenous reservoirs may give insight into methods of using generally undesirable physical phenomena in a computationally useful way.

The major finding of the SGT study showed that the computational capability of RBNs was maximized for networks with critical dynamics. The results were not obtained analytically, but reinforced previous analytic claims through the use of simulation. The simulator used to generate their results was likely a tool called *RBN Toolbox* created in part by Christof Teuscher [8].

SGT uses measures first introduced in [21] called separation, fading memory and computational capability. These measures are all based on the idea of normalized Hamming distance. The measure of computational capability is especially important to our research, as we look to extend the work of SGT and show how computational capability relates to energy dissipation.

SGT uses some previously defined measures to analyze different metrics of RBNs. These include a distance measurement between two networks known as *Hamming Distance* and a measure of a network's ability to perform useful computation called *Computational Capability*. The computational capability measure is also dependent on the definition of three other measures, *Separation*, *Fading Memory* and *Perturbance Spreading*. What follows is a detailed analysis into the specifics of how these measure are actually calculated:

Hamming Distance

The Hamming distance measure gives a metric on the distance between two states of the same network at different periods of time and possibly different input strings (if being used in a RC device). The measure counts the number of differing node outputs between the network at time t_1 and time t_2 and is defined as follows:

$$H(A, B) = \sum_i^N |a_i - b_i|$$

where $H(A, B)$ is the Hamming distance between two RBN states $\{A = S(t_1)\}$ and B$\{B = S(t_2)\}$ of the same RBN. The indices on a and b represent a specific node i for each state where a_i represents the ith node from the RBN in state A [8]. There is a common alteration on this measure called the *Normalized Hamming Distance* which is defined as:

$$H_{normal}(A, B) = \frac{1}{N} \sum_i^N |a_i - b_i|.$$

With a variation to notation, this measure is also equivalent to another measure called *Perturbance Spreading*, which is defined as $P(\mathcal{M}, u_a, u_b) = H_{normal}(A, B)$ where $\mathcal{M}$ is the RBN, and A and B are the network states after being driven by u_a and u_b respectively [4]. With this measure it is possible to see how distant two networks become with a separation in time, but independent of the network size, N. The measure of perturbance spreading is a requirement for computational capability.

Computational Capability and its Dependencies

The measure of computational capability was first presented in [21] as a method of measuring a network's capacity to perform useful computation. The theory behind this measure comes from the idea that any system capable of performing useful computation must meet a set of physical requirements, namely, the ability to remember and the ability to forget. Therefore,

built into the definition of computational capability are two other measures, *Fading Memory* and *Separation*.

Fading Memory

Fading memory measures how well an RC device can forget past perturbations. It is defined as

$$F(M,T) = P(M,u,w), \tag{15}$$

where $T = \|u\| = \|w\|$ and

$$w_i = \begin{cases} \overline{u}_i & \text{if i} = 0 \\ u_i & \text{otherwise} \end{cases}$$

where $\mathcal{M}$ is the RBN and T is the length of an input string perturbing $\mathcal{M}$.

Separation

Separation measures how well an RC device can *separate* an RBN after being perturbed by an input string differing by only the first $T - \tau$ time steps. It is defined as

$$S_\tau(\mathcal{M},T) = P(\mathcal{M},u,v), \tag{16}$$

where $T = \|u\| = \|v\|$ and

$$v_i = \begin{cases} \overline{u}_i & \text{if i} < T - \tau \\ u_i & \text{otherwise} \end{cases}$$

where $\mathcal{M}$ is the RBN, T is the length of an input string perturbing $\mathcal{M}$, and τ is the number of similar time steps at the end of the input string.

Computational Capability

With these defined, computational capability is defined as how well an RC-RBN can separate input strings minus how well it can forget. Intuitively, if an RBN can separate input strings well and forget the initial perturbation by the time the input string completes, then these properties match those of similar models with high computational abilities. Formally, the measure is defined as

$$\triangle(\mathcal{M}, T, \tau) = S_\tau(\mathcal{M}, T) - F(\mathcal{M}, T). \tag{17}$$

In this study, SGT simulates RC-RBNs with different parameters. They show that the computational capability of RC-RBNs is maximized at $<K> = 2$ which generally classifies the RBN reservoir dynamics as critical. They also mention that a high $\triangle$ signifies that the reservoir dynamics have the ability to separate differing input strings and are mostly determined by recent perturbations.

Through our research we noticed that this measure suffers from flaws which can occur on a case by case basis. As an example, the use of this measure should result in a quantity $\mathcal{Q}$ where $\mathcal{Q} \in [0, 1]$, but it can be shown that when applied to certain RC-RBNs with certain input strings, $\mathcal{Q} \notin [0, 1]$. Fortunately, when averaged over many networks, the measure tends to be a good estimator of the properties it claims to capture. Although the issue is not addressed in our work, research should be conducted in the development of new measures which do not suffer these flaws. In our research though, the chosen measures, specifically computational capability, work well in representing the dynamics of many generated networks as the averaged measure becomes more accurate as the sample count increases.

Although the SGT work published results on computational capability of RC-RBNs and the CRK work published results on the power efficiency of classical RBNs, more work needs to be done in identifying links between criticality, energy efficiency and computational efficiency of these networks. As of this writing, we know of no published research which directly relates

the measure of computational capability with measures of energy efficiency. We also do not know of any published research which analyzes how changes in parameters in an RC-RBN affect the equivalent FSA and vice versa.

1.3 Objectives

In the following section, we reiterate the motivation for our work and explain our primary and secondary research goals.

Research has shown positive results in the use of physical RC devices in machine learning applications such as speech-recognition and time-series prediction [3]. Although these results are positive, the majority of RC research has been dedicated to RNNs and other networks which homogenous in-degrees. Unfortunately, for those interested in applying RC methods to physical and self-assembled systems, the reservoir in-degrees have some level of heterogeneity due to uncontrollable design variation [4]. Despite the fact few researchers have developed the idea of RC devices with heterogenous reservoirs, many have studied the general properties of classical RBNs. Such studies include those relating to power efficiency, dynamical network properties, mean-field evaluation and many others [4, 9, 7]. This wealth of research on a heterogenous capable RC device candidate influenced SGT to evaluate RC devices with RBNs. Now though, with the added complexity involved in using the RBN as a reservoir, it is unclear which properties of classical RBNs translate into the new RC-RBN framework. This gap in literature, along with the prospect of developing a model for physical and self assembling architectures, is the primary motivation for both our research and the research of SGT.

With this motivation in mind and knowledge of previous work done in the field, we hope to show which RC-RBN parameter values maximize both computational capability and minimize average energy dissipation per time step. Therefore, our primary objective is to extend the work done by SGT [4] and CRK [9] by showing the relationship between computational capability [21] of RC-RBNs and the lower bound on average energy dissipated per time step

[10] of their equivalent IFSA. Note that there is a subtle difference in our motivation and in our primary goal: We look to show the relationship between computational capability and the *lower bound* on average anergy dissipated per time step, which will give a fundamental relationship between computational capability and energy dissipation as compared to a comparison of computational capability and the average energy dissipation of some physical implementation. In other words, the energy dissipation becomes a function of implementation, but our results remain fixed only for the *best* possible implementation (in terms of the RC-RBN model presented in Sec. 1.1.2 and irreversible information loss).

Through the development of our research methodology, driven by our primary goals, we have encountered a number of questions which now serve as motivating factors in our secondary objectives. As has been explored by both CRK and SGT, RBNs have the tendency to behave similar to physical phenomena when their dynamics are critical. On average, for classical RBNs modified for heterogenous in-degrees, this phase occurs as a function of the parameters, $\{N, <K>, p\}$ such that when $p = 0.5$, the critical phase occurs at $<K> = 2$. Therefore, we explicitly pay close attention to results which occur near the critical phase. We hope to show which (if any) properties of an RC-RBN's equivalent FSA are maximized for $<K> = 2$.

As we progressed in our research, we quickly ran into issues (details in Sec. 2.6) with the sheer size of the equivalent FSA and the computation time in iterating the depth-first search tree required to prove irreducibility (details in Sec. 2.2). This led to the inevitable conclusion that we would have to study only networks with fewer than 15 nodes (*Small-N networks*). Yet, while exploring this restrictive set of networks, we realized it was possible to extract IS-FSA from the larger FSA (details in Sec. 1.1.3). This led to further questions which served as motivation for additional objectives, namely, how do the properties of RC-RBNs map to IS-FSA? Through our research we hope to show the relationships which exist between RC-RBNs and IS-FSA, and the relationships between IFSA and IS-FSA.

2 Methodology

With our research concepts, language and objectives defined, we now provide a complete discussion of our research methodology. This section provides an overview of our Java simulation suite, an analysis of developed algorithms for finding IFSA and IS-FSA, an explanation in the confidence we have in our pseudorandom number generator, an explanation of statistical reliability which includes our Java unit tests and a discussion of issues we faced in developing our methodology. All of these sections work to instill trust in our simulation suite while showing the features critical to the generation of our results.

2.1 Simulation Overview

This research required not just an understanding of RC, RBNs, and FSA, but it also required quite a bit of programming to create a suite of simulation tools with which to conduct our research. Although some tools already exist, we decided to develop our own to help promote a deeper understanding and give finer control over how we perform simulation. This control allowed us to add all the previously mentioned measures to the simulator. Currently our simulator has the following features:

RC-RBN

- Write to File: Exports an RC-RBN to a comma-separated value (CSV) file which stores all of the network parameters and the randomly selected inputs and internal connections.
- Load from File: Reads an RC-RBN from a comma-separated value (CSV) file which was written with the "Write to File" function and generates a new instance of that RC-RBN.
- Hamming Distance: Calculates the Hamming distance of an RC-RBN instance after being run with two different input strings.

- Perturbance Spreading: Calculates the perturbance spreading of an RC-RBN instance after being run with two different input strings.
- Separation: Calculates the separation of an RC-RBN instance with a given input string and a given value for τ.
- Fading Memory: Calculates the fading memory of an RC-RBN instance with a given input string.
- Computational Capability: Calculates the computational capability of an RC-RBN instance with a given input string and a given value for τ.
- Run Input String: Simulates the RC-RBN instance with a given input string. The RC-RBN will sequentially update its state until all bits in the input string have been processed.
- Print State Space Map: Generates a chart where each row represents the state of the network at a particular time step.
- Print Change Map: Generates a chart where each row represents the change in the state of the network at a particular time step, such that each column gives the outputs of a particular node.
- Inspect Internal State (Dec / Bin): Returns the internal state of the given RC-RBN instance. This function is overloaded to return either a decimal representation or a binary array.
- Regenerate Internal Nodes: Regenerates the random connections internal to the RBN reservoir while leaving the input connections as they were.

FSA

- Write to File: Exports an FSA to a *comma-separated value (CSV)* file which stores the states of the FSA and their transition rules.

- Load from File: Reads an FSA from a CSV file which was written with the "Write to File" function and generates a new instance of that FSA.
- Find IS-FSA Given State (Algorithm / Matrix): Finds and returns the IS-FSA object for the given state if, and only if, it exists. There are two functions to perform this action, where one uses an algorithmic approach and the other uses a matrix decomposition.
- Find All IS-FSA (Algorithm / Matrix): Finds and returns a list of all the IS-FSA objects which can be located in a given FSA. There are two functions to perform this action, where one uses an algorithmic approach and the other uses a matrix decomposition.
- Run on Input String: Simulates the FSA instance with a given input string. The FSA will sequentially update its state until all bits in the input string have been processed.
- Show Visualization: Generates a visual representation of an FSA instance.
- Show Visualization with IS-FSA: Generates a visual representation of an FSA instance which highlights nodes that are included in IS-FSA.
- Convert to Matrix: Converts an FSA instance to a probability transition matrix.
- Test for Irreducibility: Returns a *Boolean* value reporting whether or not an FSA instance is in fact an IFSA.
- Get Energy Dissipation (only IFSA): Returns a *Double* value corresponding to the lower bound in average energy dissipated per time step in units of $(K_B T ln(2))$.

RC-RBN / FSA

- RC-RBN to FSA Conversion: Converts an instance of an RC-RBN into an equivalent FSA.

- Show RC-RBN Visualization: Shows a visualization of an RC-RBN instance.

Simulation

- Batch Simulator: This is a key feature of the simulator. The batch simulator can be set up to dynamically simulate any ordering of RC-RBN parameters swept over a given range. Upon completion, the simulator will generate a CSV file which includes both the parameter values, and the results of the simulation. There are a number of different simulations which can be performed.

- 2D Plot from CSV: Generates a 2D chart from a CSV file which was generated by the batch simulator.

- 3D Plot from CSV: Generates a 3D chart from a CSV file which was generated by the batch simulator.

- Generate Heat Map: Generates a 2D heat map from a CSV file which was generated by the batch simulator.

Helping Functions / Other

- Shannon Source Generator: A class dedicated to supplying output to emulate a typical IID binary source.

- High Reliability Random Generator: A random class which ensures a high reliability uniform distribution (see 2.4).

- String to Int Array: Converts a string of binary characters to a an array of integers.

- Dec to Bin Array: Converts a integer into an integer array holding its binary representation given a certain number of binary digits.

- Bin Array to Dec: Converts an iteger array holding a binary representation into an equivalent integer.

- Statistic Manager: Keeps track of averages for the batch simulator.

IFSA Manager

- Batch Export IFSA mapping RC-RBNs: Randomly generates RC-RBNs and saves networks which map to IFSA using the RC-RBN "Write to File" function.

- Import IFSA Mapping RBNs as RC-RBNs: Reads IFSA maping RC-RBNs from files which were saved with the "Batch Export IFSA mapping RC-RBNs" function.

Of these features, the most notable is the ability to convert an arbitrary RC-RBN to an equivalent FSA. The motivation behind the development of this conversion tool was the requirement set in [10] for the use of IFSA in determining the minimum average bound on energy dissipated per time step. This tool was extended to another piece of software called the *IFSA Manager* which controls the importing and exporting of IFSA mapping RC-RBNs.

2.2 Testing for Irreducibility

The ability to differentiate an IFSA from a general FSA is an important tool in our research, and we were able to accomplish this goal using two different methods. The first is an algorithmic approach and the second uses a standard Eigenvector / Eigenvalue decomposition library. Alg. 4 shows how to pragmatically test an FSA for irreducibility, and it is located in the Appendix.

In [10], GA explains that it is possible to determine whether an FSA is irreducible through its probability transition matrix alone. Given an FSA, $\mathcal{F}$, with transition matrix P, $\mathcal{F}$ is irreducible if, and only if, there exists a unique Eigenvector π (Occupation probability vector) such that:

- $\pi = \pi P$
- Every element in π is non-zero [10].

This method first converts the FSA into a probability transition matrix. Once this matrix is built, it is transposed so the decomposition will give left-hand Eigenvector / Eigenvalue pairs. Eigenvectors which have an Eigenvalue of *1* are extracted and if this vector list has only one vector, then it is checked for 0s. If no 0s exist, then the FSA is irreducible. In programming the actual method, the *JAMA* library was used. This library contains support for all the basic matrix functionality and also supplied some basic decompositions. The algorithm for this matrix method of determining irreducibility can be seen in the Appendix under Alg. 3.

In our second method in proving irreducibility, we use three functions to fill a matrix with either 1s or 0s representing the ability for a state to reach other states. If the output of the algorithm is a matrix filled with 1s, then the FSA is irreducible. The algorithm we developed can be found in the Appendix under Alg. 4. In our research, the second method has been a much faster alternative to the matrix decomposition method.

2.3 Irreducible-Sub FSA

Although not required in reaching our primary objective, the ability to find IS-FSA is important in extending our current work from IFSA to general FSA. This allows for the study of all RC-RBNs, and not just IFSA mapping RBNs. The ability to locate these state sets is important as these are the steady states of the general FSA. In other words, these state sets each correspond to an Eigenvector with an Eigenvalue of 1. The normalized Eigenvector represents the steady state occupation vector for the state set. To actually find these state sets we use an algorithmic aproach. The specific algorithm can be found in the Appendix under Alg. 5.

2.4 Random Number Generation

One critical component of the Java RBN simulator is the software responsible for providing a reliable *pseudorandom number generator (PRNG)*. The PRNG should be capable of

constantly drawing from a uniformly random distribution. To do this, a custom class was created called *CustomRandom*, which is a class wrapper around Java's *Random* class. It is documented that the Java *Random* class can produce 2^{48} random numbers before restarting its cycle [22]. The *CustomRandom* class is used throughout the entire program and it regulates the usage of random numbers. As soon as 2^{48} numbers are generated, the internal *Random* instance object of the *CustomRandom* class is reinitialized with the default constructor. Fortunately, it is unlikely this feature will be required as the simulator will have to generate over 281 trillion numbers before this feature is activated. Regardless, we have tested the general capability of this PRNG by generating 10 Billion numbers between zero and one hundred. We initialized an array of size 101, and added *1* to each array slot (bucket) every time the PRNG hit that number. This histogram is plotted in Fig. 4.

Given a perfect uniform distribution, the average bucket error would be 0%. Instead, the data from the simulator shows an average bucket error of 0.028%. This measure shows that the Java PRNG is generating numbers as if pulled from a uniform distribution, and the sampled distribution shows evidence that this PRNG *is* suitable for our purposes.

2.5 Statistical Reliability

In any statistical study the results are only as good as the validity of the samples. So, we have made a point to employ a set of unit tests in our simulator to ensure the code is functioning properly. In this section, we give an overview of all the simulation tests.

Simulator Unit Tests

- *Hamming Distance:* Ensures the hamming distance measure is working properly. In this test, the strings "0011" and "0101" are compared, and the test succeeds if the function returns 2.

- *Input String One:* Tests the RBN simulator by loading and running the RBN in Fig. 5. After initializing in state 0 and running this network on input string "0011" the

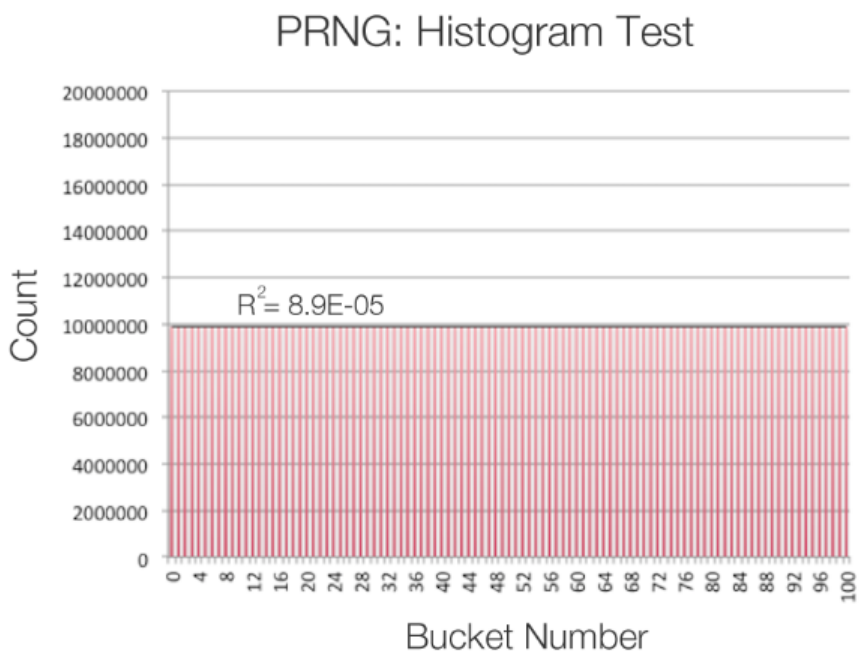

Figure 4: The purpose of this test is to show that the pseudorandom number generator in our simulator will work for our purposes. The figure shows a histogram of "buckets" which our pseudorandom number generator used to sort 10 billions samples from a uniform distribution of numbers ranging from 1 and 100. Due to the small network sizes in our simulation, the range of numbers we will be generating is even smaller than 1 to 100, so this is not a conservative example.

state of the network should be "011".

- *Input String Two:* Tests the RBN simulator by loading and running the RBN in Fig. 5. After initializing in state 0 and running this network on input string "0101" the state of the network should be "010". This is the same as the previous test, but put in place for some redundancy.

- *Perturbance Spreading:* Tests the perturbation spreading measure using the RBN shown in Fig. 5. The test calculates the perturbance spreading of input string "0011" and "0101". If the result is 1/3 then the test passes.

- *Separation:* Ensures the separation measure is working properly. This is done by using the input string "0101" with $\tau = 1$. This test uses the FSA in Fig. 5. In this case the test passes if the calculated separation is 0.

- *Irreducibility Test - Algorithm - Success:* Tests the validity of the algorithmic method of checking for irreducibility. This test uses the FSA in Fig. 6 . Since this FSA is irreducible, the it isIrreducibleViaAlgorithm method needs to return *True*.

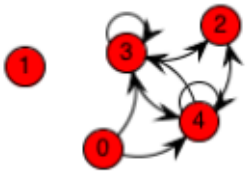

Figure 5: Random test RBN

Figure 6: Irreducible FSA

- *Irreducibility Test - Algorithm - Failure:* Tests the validity of the algorithmic method of checking for irreducibility. This test uses the FSA in Fig. 7 . Since this FSA is not irreducible, the *isIrreducibleViaAlgorithm* method needs to return *False*.

- *Irreducibility Test - Eigenvalue / Eigenvector - Success:* Tests the validity of the eigenvalue / eigenvector method of checking for irreducibility. This test uses the FSA in Fig. 6 . Since this FSA is irreducible, the *isIrreducible* method needs to return *True*.

- *Irreducibility Test - Eigenvalue / Eigenvector - Failure:* Tests the validity of the eigenvalue / eigenvector method of checking for irreducibility. This test uses the FSA in Fig. 7 . Since this FSA is not irreducible, the *isIrreducible* method needs to return *False*.

- *Energy Dissipation - Reversible:* Tests the validity of the energy dissipation functions for the reversible FSA shown in Fig. 8. Since the FSA is reversible, there is no requirement for energy dissipation.

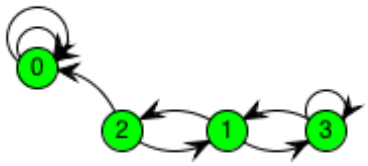

Figure 7: Non Irreducible FSA

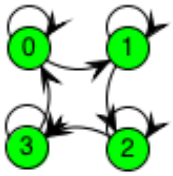

Figure 8: Reversible FSA

- *Energy Dissipation - GA Example:* Tests the validity of the energy dissipation functions for the FSA presented in [10] and shown in Fig. 9. Since the FSA is irreversible, there is a specific average minimum bound on the energy dissipated per time step. When fed by a IID source which generates a "1" 50% of the time, there should be about 0.82011 bits of information lost per state transition.

- *IS-FSA Discovery - Algorithm:* Tests the validity of the algorithmic IS-FSA discovery code. Specifically, the test uses the FSA from Fig. 9 to check which IS-FSA state "2" is in. In this case, the IS-FSA should be the entire network.

- *IS-FSA Discovery - Algorithm - Circuit of One:* Tests the validity of the algorithmic IS-FSA discovery code for an FSA state which is a circuit of one (CO1) or single state sub FSA. The test uses the FSA from Fig. 7 to check the size of the IS-FSA state "0" is in. Since it is a CO1 IS-FSA, the results should be "1".

- *IS-FSA Discovery - Algorithm - CO3:* Tests the validity of the algorithmic IS-FSA discovery code for an FSA state which is in a CO3 IS-FSA. The test uses the FSA from Fig. 10 to check the size of the IS-FSA state "1" is in. Since it is a CO3 IS-FSA, the results should be "3".

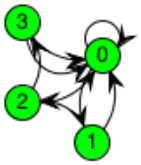

Figure 9: GA Example FSA

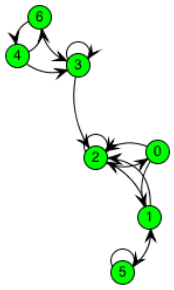

Figure 10: CO3 Attractor FSA

- *All IS-FSA Discovery - Algorithm:* Tests the abilities of the *getAllISFSA* function. In this case, we need to ensure the function has successfully extracted all the IS-FSA from the general FSA. The test uses the FSA shown in Fig. 11 to check if the size of the returned list of IS-FSA is "3".

- *FSA - RBN Equivalence Test:* Tests the simulator's ability to simulate both an instance of an RBN and its equivalent FSA. The test ensures that both end in the same state after being driven by the same input string. The RBN is generated randomly for each testing instance, and the equivalent FSA is computed. Then, a random input string is generated by an IID source with a length of 1,000 and a 50% chance of a 1. Both the RBN and FSA are run with the input string and their states are directly compared. If they are equivalent, then the test succeeds.

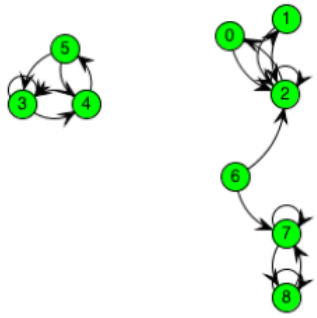

Figure 11: 3 Attractors FSA

2.6 Simulator Development Status

As we progressed in our research we came across quite a few different issues, some of which we did not anticipate and still need to be overcome. Other issues we were able to successfully resolve. This study has required intensive programming, and like any large programming project, the only way to fix a problem is to focus and debug for as long as it takes. In the next two sections, we list some of the main issues we had while developing the RBN simulation software.

2.6.1 Resolved Issues

- The *Efficient Java Matrix Library (EJML)* was our first choice of matrix library, and unfortunately we were unable to successfully decompose matrices which lead to instances of non-real Eigenvalues. To fix this issue we switched to a different matrix library called *Java Matrix Package (JAMA)*.

- We had an issue with extremely long simulation times for finding IS-FSA. This was fixed using an algorithmic approach instead of a matrix decomposition approach. The algorithm was also optimized by using previously attained data to perform a depth first search on the FSA graph.

- There was some difficulty in the way the simulator was set up in terms of arrays and node ordering. To overcome these issues we have written some documentation which clearly explains how networks are read / written to and from files and how the program expects to see the data.

- There were issues in developing the depth first search code to find all attractors within a given FSA. The issues were mostly in the design of the algorithm and its complexity.

- We had quite a bit of trouble trying to make a robust batch simulator. In the end, we were able to create multiple discrete variable objects which were then used to

pragmatically choose which simulation to run.

- Finding ways to generate nice looking charts from Java was actually more difficult than we ever would have expected. Although we found a few libraries, we had to do some manual tweaking for the desired results.

2.6.2 Unresolved Issues

The following list outlines some issues we had which were unable to be resolved. Although these are unresolved issues, they in no way hindered the results of our simulations, although they are obstacles which need to be overcome to perform simulations with larger RC-RBNs.

- Although we changed our matrix library to the JAMA library, we are still seeing issues in the decomposition function for large matrices. Eventually, the library gets stuck processing a certain matrix and does not finish. On top of this, the library is slow for large matrices.

- Even when using the algorithm, the time it takes to discover IS-FSA for anything but *Small-N* networks is too long to perform practical simulations for larger sized N. The reason for this time is simply the size of the search tree. Since there are 2^N states, there are 2^{2^N} paths in the search tree (not including the optimizations we have already done).

- It is difficult to convert RBNs with $N > 25$ as the size of the FSA is too large to be held in memory.

- There is an issue with the code which does a step by step transversal of the RC-RBN, as to compute this energy dissipation we need to know the states which could have led to any given state using only the wiring diagram.

- There is still no chart utility to generate a 3D chart which has colored vertices (think Matlab) to show depth. Although, we have a library in mind, it will take some time

to implement.

3 Exploring Properties of RC-RBN, IFSA and IS-FSA

We now present the results of our study on the relationships between computational capability and energy dissipation, criticality and properties of equivalent FSA and the relationships between properties of RC-RBNs, IFSA and IS-FSA. Throughout this section we pay close attention to results where some property is maximized or minimized at $<K> = 2$ as this implies criticality with our simulation setup of $p = 0.5$ (explained in Sec. 1.2) [14].

This section is split into four different subsections, each of which presents a specific set of studies related to our objectives. In Sec. 3.1 we investigate the probability of generating IFSA mapping RC-RBNs which is a requirement in applying the lower bound on energy dissipation. In Sec. 3.2 we present our primary study and show the relationship which exists between computational capability and the lower bound on average energy dissipated per time step for IFSA mapping RC-RBNs.

In the last two sections (Sec. 3.3 and 3.4) we explore alternatives to the use of IFSA mapping RC-RBNs in developing a minimum bound on the average energy dissipated per time step. Specifically, in Sec. 3.3 we present a statistical study of how RC-RBNs map to IS-FSA to better understand how IS-FSA may be used as an alternative to IFSA in studies of energy dissipation. In Sec. 3.4 we give a preliminary study for energy dissipation using IS-FSA.

3.1 Exploration on Irreducibility

When we first began our study, we quickly realized that we were limited by the scope of our energy dissipation measure, as it is only applicable to IFSA. As such, we proceeded with caution, as any study done with this measure would only be applicable to RC-RBNs which map to IFSA. Due to the statistically representative nature of simulatory studies, we

decided to perform a preliminary experiment to determine how likely it is that generating an instance of a RC-RBN would result in an IFSA mapping RC-RBN.

There are three important parameters in the study of RC-RBNs, and these are $\{N, <K>, L\}$ as defined in Sec. 1.1.1 and 1.1.2. We performed two different tests, each of which show how changes in the RC-RBN parameters relate to differences in the probability of generating RC-RBN mapping IFSA. The first simulation gave results for the calculated probability of generating IFSA mapping RC-RBNs with $<K> \in \{1, 2, 3\}$, $N \in \{3, 4, 5\}$ and $L = N$, averaged over 50,000 samples per N. The choice to set L equal to N comes from previous research in [4], where computational capability is shown to be higher for values of L closer to N (the network is being perturbed more). This means the results of our first test show the calculated probability that an instance of an RC-RBN will map to an IFSA while keeping properties well suited for high computational capability. One may notice that the range of N is limited in this study. The reason for this is simply the computation time required to process such a high number of samples for an accurate representation of IFSA probability. Also, it should be noted that for all our studies, N is lower bounded such that $N>3$ because we always simulate with $<K> \in [1, 3]$. The results of this test can be seen in Fig. 12 and 13. The first shows the data plotted on a linear scale while the second is plotted on a logarithmic scale.

Given the results shown in Fig. 12 and 13, it is not difficult to see that the number of IFSA mapping RC-RBNs decreases exponentially as N increases. Once again, this is another reason our study only focuses on *Small-N* networks. Also, note how the probability for $<K> = 3$ starts high, and then descends through the other two values of $<K>$. This phenomenon will become a trend throughout our remaining results.

Although we have seen evidence that the probability of generating IFSA mapping RC-RBNs decreases exponentially for increasing N, we still cannot say anything about its relationship to the number of input connections, L. So, in the next simulation we sweep through $<K>$, N and L to see how the number of nodes connected to the input layer affects the

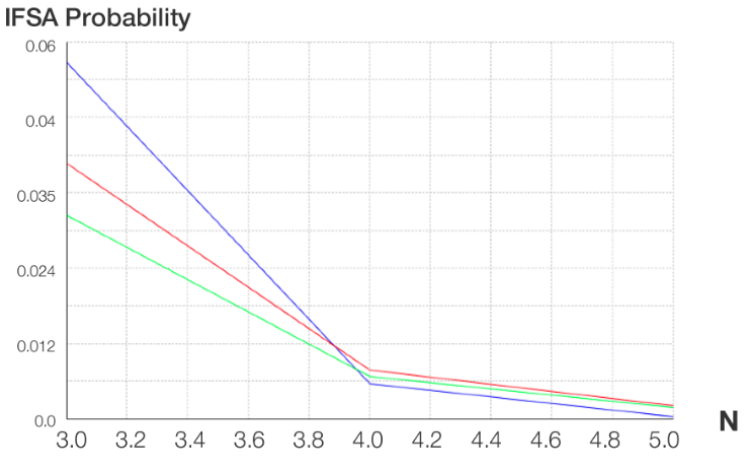

Figure 12: Blue: $<K>=3$, Red: $<K>=2$, Green: $<K>=1$
As a preliminary study to calculating the energy dissipation bound, we show the probability of generating IFSA mapping RC-RBNs with $<K> \in \{1,2,3\}$, $N \in \{3,4,5\}$ and $L=N$, averaged over 50,000 samples per N. These values were chosen to maximize computational capability. This plot is shown with a linear scale.

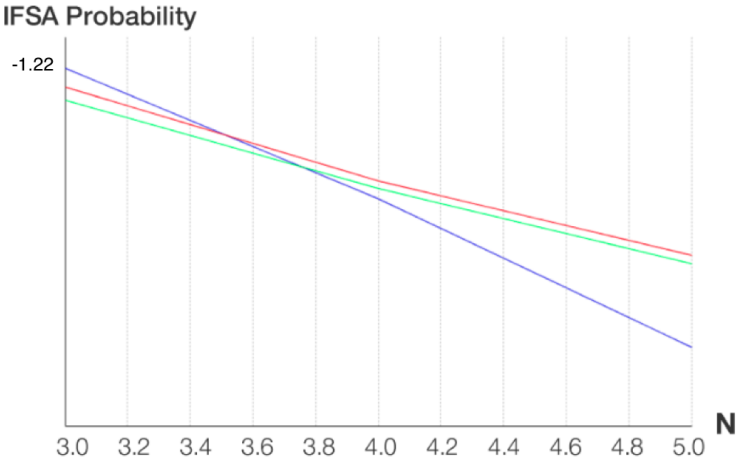

Figure 13: Blue: $<K>=3$, Red: $<K>=2$, Green: $<K>=1$
As a preliminary study to calculating the energy dissipation bound, we show the probability of generating IFSA mapping RC-RBNs with $<K> \in \{1,2,3\}$, $N \in \{3,4,5\}$ and $L=N$, averaged over 50,000 samples per N. These values were chosen to maximize computational capability. This plot is shown with a logarithmic scale.

probability of generating IFSA mapping RC-RBNs. The tests were run with $<K> \in [1, 3]$, $N \in [4, 7]$ and $L \in [1, 4]$ averaged over 10,000 samples per N. The results are shown in Fig. 14, and each heat map shows a different value of $<K>$.

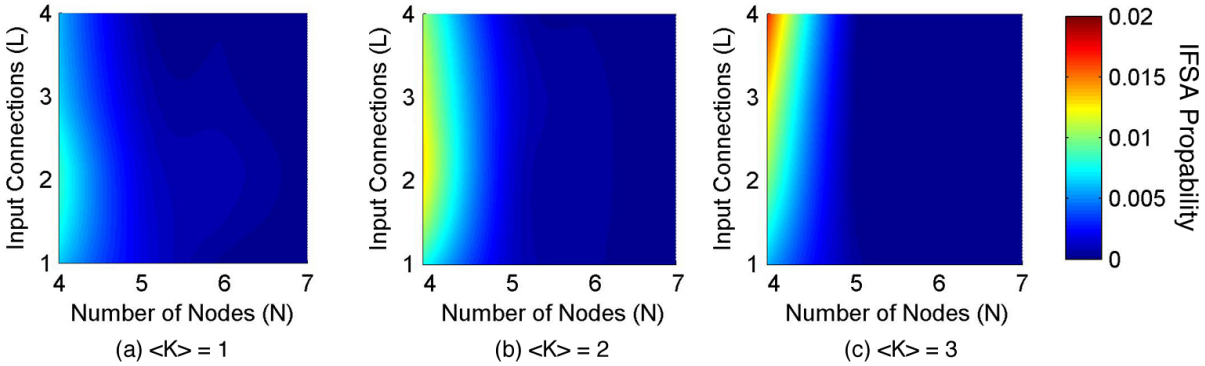

Figure 14: Results of a simulation performed to study the probability of generating an IFSA mapping RC-RBN with the following parameter space: $<K> \in [1, 3]$, $N \in [4, 7]$ and $L \in [1, 4]$ averaged over 10,000 samples per N. These heat maps show how L impacts the probability of generating an IFSA *(cubic interpolation of discrete data)*.

Looking at the results, there is a noticeable noticeable trend in L corresponding to the $<K> = 3$ plot (plot c). Yet, plots a and b do not support the claim that IFSA probability increases with L. On the other hand, all three plots agree that IFSA probability decreases exponentially with N. The plots also show the strange case where $<K> = 3$ rises above the other connectivities when N is small. It is clear that IFSA probability decreases with N, and because of this, the remainder of our study focuses on *Small-N* RBNs.

3.2 Computational Capability and Energy Dissipation of Small-N RC-RBNs

Through this research we sought a relationship between computational capability and the defined measure of energy efficiency. To reiterate, we employ the use of the computational capability measure first introduced in [21] and the energy dissipation bound introduced in [10]. SGT shows in [4] that the computational capability measure gives a quantity, $\mathcal{Q} \in [0, 1]$ which

determines how well an RC-RBN with given parameters can compute information when averaged over many instances. In [10], GA presents a theorem which gives a fundamental lower bound on the average energy dissipated per state transition of an IFSA.

The previous section (Sec. 3.1) gave proof that it would be increasingly difficult to generate an IFSA mapping RC-RBN as the number of nodes in the RBN (N) increased. So, we limit this study to RBNs with fewer than 15 nodes (*Small-N* RBNs). For this reason, we ran a test to see if the computational capability of *Small-N* RBNs was similar to that of larger networks. To perform this test, we ran the simulation with the following parameter space: $<K> \in [1, 3]$, $L = N(0.4)$, $T = 5$, $\tau = 2$ and $N \in [5, 60]$ averaged over 1,500 samples per N. Our motivation in choosing $L = N(0.4)$ also came from the results of SGT. It can be seen that when $L \approx N(0.4)$ there is a clear difference in the computational capability between the different values of $<K>$. The research of SGT showed that as L approaches N, the computational capability begins to rise to the level close to $<K> = 2$. So, in setting L such that it shows separation in the levels of criticality we hoped to show that high values of computational capability could be reached with a low number of input connections. The results of this study can be seen in Fig. 15

The results from this test are interesting for a few different reasons. First, it is clear that computational capability does not follow the expected trend (results of SGT [4]) for *Small-N* RBNs, as the computational capability of chaotic connectivity ($<K> = 3$) starts high and drops down and converges with values of ordered connectivity ($<K> = 1$). Second, the behavior of $<K> = 3$ matches closely with the behavior seen in Fig. 13. Once again, a phenomenon which will continue throughout our results.

Next, we were finally prepared to perform our study of computational capability vs energy dissipation for IFSA mapping *Small-N* RC-RBNs. To perform this study, we first generated 50 instances of RC-RBNs which mapped to IFSA for the following parameter set: $<K> \in [1, 3]$, $N \in [3, 7]$ and $L = N$. After pulling these networks from randomly generated RC-RBNs, we applied our selected measures of computational capability and energy dissipation,

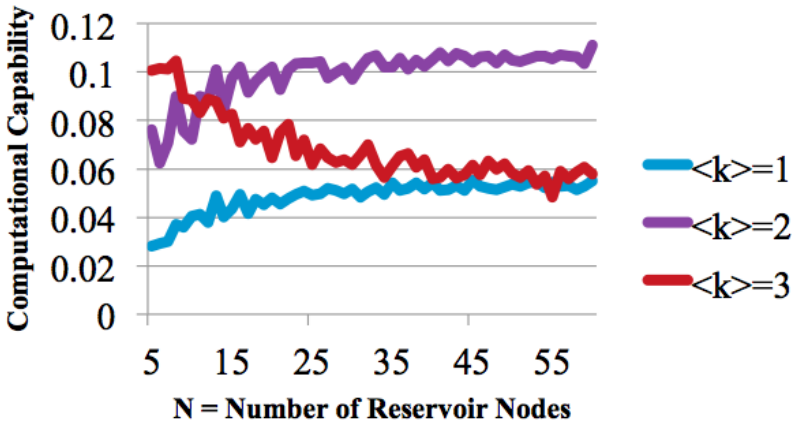

Figure 15: To see the transition between *Small-N* networks and larger networks, this figure shows the computational capability of networks with the following parameter space: $L = N(0.4)$, $T = 5$, $\tau = 2$, $N \in [5, 60]$ averaged over 1,500 samples. These values were chosen to show maximum variation in the computational capability at different phases.

and plotted these to get the correlation plot shown in Fig. 16.

Although these results support the idea of a predictive relationship between computational capability and energy dissipation, there is not enough evidence to make a claim, because the behavior of *Small-N* RC-RBNs differs from that of RC-RBNs with more than 15 nodes. Regardless, the clear trend of the correlation plot is intriguing. The outliers may only exist in *Small-N* RBNs because of the chaotic plot line. As a soft conclusion, the restriction of using IFSA mapping RC-RBNs makes this statistically representative study nearly impossible. This is our motivation to extend the bound to more than just IFSA.

3.3 Extending the Bound - Statistics of IS-FSA

Because RC-RBN mapping IFSA are only a small subset of all possible RBNs, we now explore IS-FSA to further our study. Although we are interested in using IS-FSA to extend the usefulness of the energy dissipation bound, we also study their properties similarly to how we studied the probability of generating IFSA mapping RC-RBNs. So, in this section, we present our studies of IS-FSA in which we show some correlation between the properties

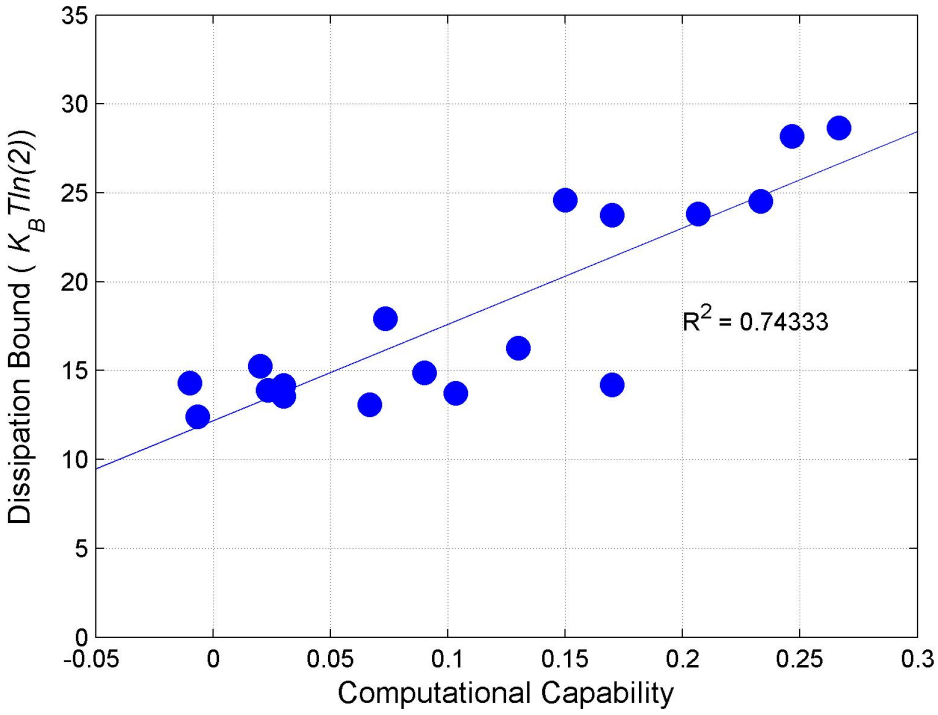

Figure 16: A correlation plot of computational capability vs the lower bound on energy dissipated per time step of networks with the following parameter space: $textless K > \in [1, 3]$, $N \in [3, 7]$ and $L = N$ with 50 samples per N. This plot gives evidence to our primary objective in showing a predictable relationship between computational capability and the lower bound on energy dissipated per time step of *Small-N* networks.

of RC-RBNs and properties of IS-FSA. This correlation may give clues to the meaning and intuition behind RC-RBN properties (such as criticality). In this section we present two studies using measures for IS-FSA which we then compare to our previous results form Sec. 3.1 and 3.2.

Before using IS-FSA for energy calculations, we decided it best to explore the IS-FSA space and show which features of IS-FSA relate to their RC-RBN counterparts. One interesting feature we study is the percentage of the states in an FSA which are involved in IS-FSA (*IS-FSA State Probability*). The motivation behind this test was simple, the higher the IS-FSA state probability, the higher the probability of reaching an IS-FSA state from a

random starting point. The study was set up as follows: $<K> \in \{1, 2, 3\}$, $N \in [5, 13]$ and $L = N$ averaged over 1,000 samples per N. The results of this test can be seen in Fig. 17 and 18 . Fig. 17 shows this data plotted on a linear scale and Fig. 18 shows the plot on a logarithmic scale.

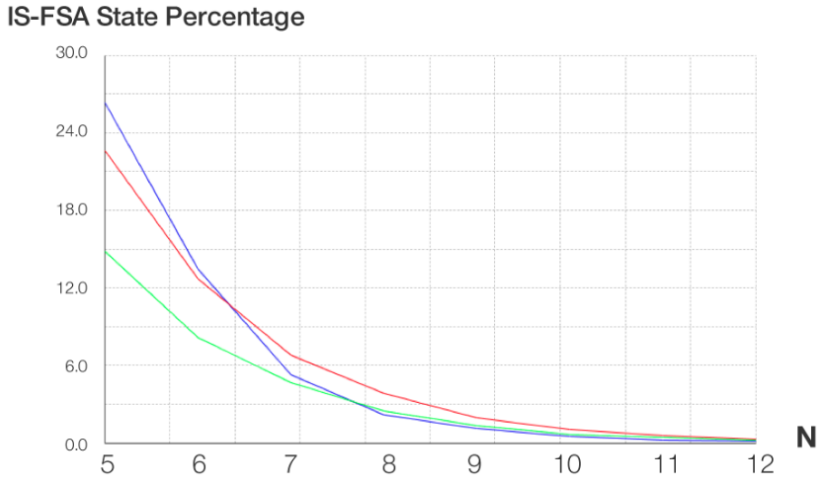

Figure 17: Blue: $<K> = 3$, Red: $<K> = 2$, Green: $<K> = 1$
As a preliminary study to calculating the energy dissipation bound with IS-FSA, we show the percentage of states in each FSA which are involved in IS-FSA with the following parameter space: $<K> \in \{1, 2, 3\}$, $N \in [5, 13]$ and $L = N$ averaged over 1,000 samples per N. This plot is shown with a linear scale.

Interestingly, these match the previous test in Fig. 12 and 13. This shows that as N increases, the percentage of states within each FSA which are involved in IS-FSA decreases exponentially. Intuitively, this makes sense, because an IS-FSA is just a generalized case of IFSA. As for the bigger picture though, these results show that as N increases, the probability of driving the state trajectory into an IS-FSA decreases.

Now, we show how the RC-RBN parameters affect the average size (in states) of each IS-FSA. Despite the fact that the percentage of IS-FSA states decreases with increasing N, the average number of states involved in each IS-FSA may increase. Our study is set up with the following parameter space: $<K> \in \{1, 2, 3\}$, $N \in [5, 12]$ and $L = N$, averaged over 5,000 samples per N. Fig. 19 shows the results of this test.

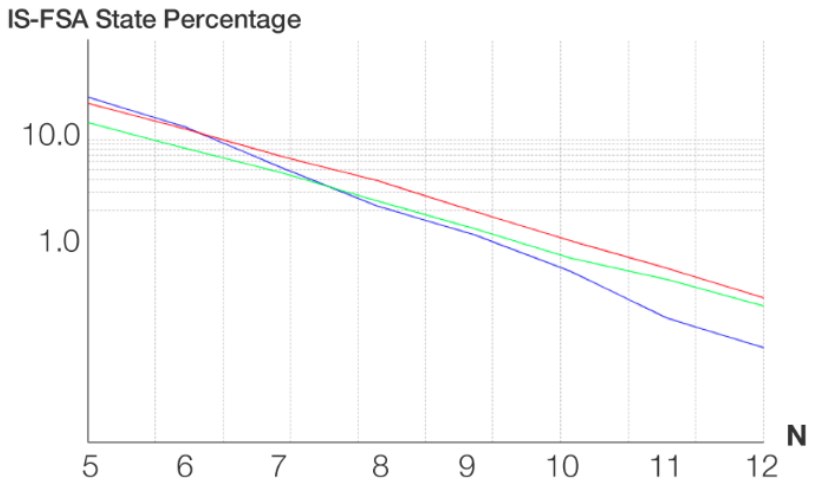

Figure 18: Blue: $<K>=3$, Red: $<K>=2$, Green: $<K>=1$
As a preliminary study to calculating the energy dissipation bound with IS-FSA, we show the percentage of states in each FSA which are involved in IS-FSA with the following parameter space: $<K> \in \{1,2,3\}$, $N \in [5,13]$ and $L=N$ averaged over 1,000 samples per N. This plot is shown with a logarithmic scale.

The results of Fig. 19 show an interesting relationship between the size of IS-FSA and the computational capability shown in Fig. 15. Looking at both plots, one can begin to see how the size of IS-FSA and the computational capability for the $<K>=3$ plot line both cross the $<K>=2$ plot line around $N=10$. More research would need to be done to verify a predictable relationship, as our current algorithm (Alg. 5) used to compute this data is on the order of: $\mathcal{O}(N^2)$ which causes any value of N>12 to take hours, up to days of computation time. Nevertheless, we have not yet shown how IS-FSA size responds to L, and therefore we perform another simulation with the following parameter space: $<K> \in [1,3]$, $N \in [5,12]$, $L \in [1,5]$ averaged over 5,000 samples. The results of this test are shown in Fig. 20.

Interestingly, the heat maps show evidence that the size of the IS-FSA increase with all three parameters, N, L and $<K>$ independently of one another. Of course, we already know from Fig. 19 that this independence stops toward the upper edge of the *Small-N* RC-RBN space. We can however claim that with the exception of $<K>=3$, IS-FSA size

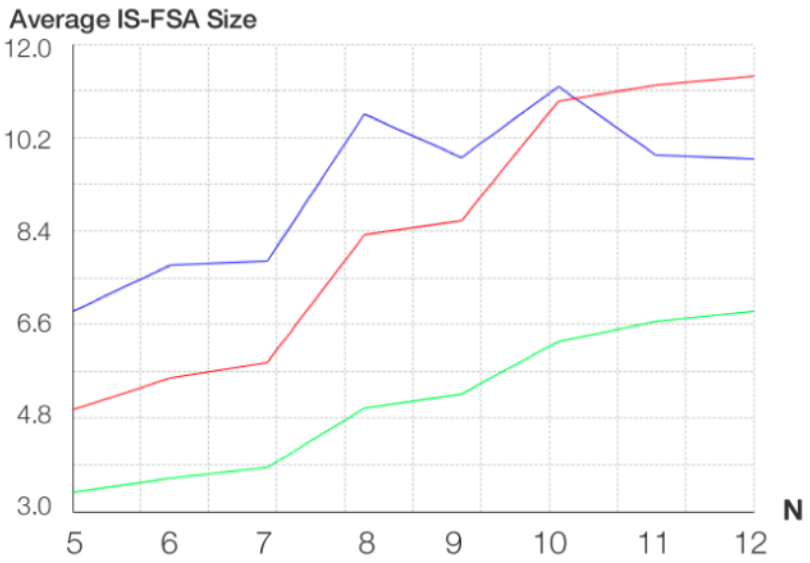

Figure 19: Blue: $<K>=3$, Red: $<K>=2$, Green: $<K>=1$
As a secondary objective, we are interested in how parameters in the RC-RBN affect the average IS-FSA size for RBNs with the following parameter space: $<K> \in \{1, 2, 3\}$, $N \in [5, 12]$ and $L = N$, averaged over 5,000 samples per N. This plot shows the average size (in states) for the IS-FSA which correspond to the given RC-RBN.

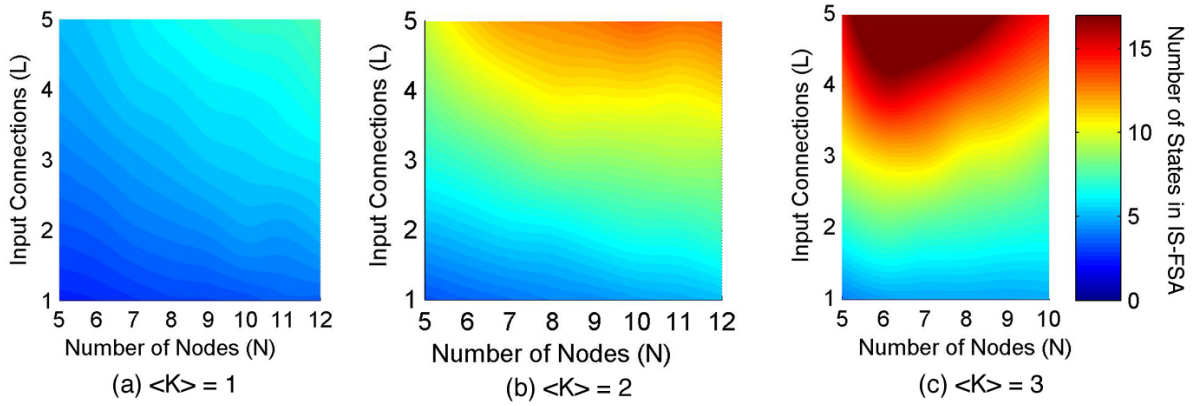

Figure 20: Three plots showing the average size of IS-FSAs for the following RC-RBN parameter space: $<K> \in [1, 3]$, $N \in [5, 12]$, $L \in [1, 5]$ averaged over 5,000 samples. The plots show evidence that the average size of IS-FSAs increase independently with N, L and $<K>$ for the lower half of the *Small-N* network space *(cubic interpolation of discrete data)*. *Note: For* $<K>=3$, $N \in [5, 10]$ *due to extensive required simulation time.*

for $<K> \in [1, 2]$ seems to increase linearly and independently for N, L and $<K>$. Once again, with more computational power, time, and algorithmic efficiency, these results could be easily extended.

3.4 Energy Dissipation in IS-FSA

With the study of IS-FSA, we no longer need to extract just networks which map to IFSA. Instead, we keep all equivalent FSA and run their state machines with a random input string of length T. Upon reaching the end of the string, we use our IS-FSA algorithm (Alg. 5) to determine whether or not the last state is part of an IS-FSA or not. If it is, then we determine which states are included, and then convert these into independent IFSA. This is possible because once an IS-FSA has been entered, is has its own steady state occupation vector, π which gives the occupation probability for each state in the IS-FSA (where occupation probability is the probability the current state is some specified state). Now, we can apply the energy bound to the extracted IFSA.

We now know that the percentage of FSA states involved in IS-FSA decrease exponentially with increasing N. While this finding may seem to complicate our study, the only real requirement for a usable sample is for the state trajectory to collide with a state within an IS-FSA out of the T length input string. Once this happens, we are guaranteed to stay in that IS-FSA indefinitely. This ability gives us not just a higher sample size but also the opportunity to extend N to the full range of our defined *Small-N* space. In this way, we investigate energy dissipation in the space where $N \in [5, 15]$. As a new measure of energy dissipation per time step, we generate an input string S_1 which has length T. After generating an instance of an RC-RBN, $\mathcal{M}$ with parameters $<K>$, N and L, S_1 is run on $\mathcal{M}$ until it reaches its end. At this point, the FSA corresponding to $\mathcal{M}$ is inspected to see if the current state is located within an IS-FSA. If it is, then the RC-RBN has reached a steady state. In this case, it is now possible to separate IS-FSA into an independent IFSA and determine the energy dissipation per time step. This is by no means a complete extension, but these

results may be representative of larger N RC-RBNs.

We now show the results of the IS-FSA study for the lower bound on average energy dissipated per time step of RC-RBNs with $<K> \in \{1, 2, 3\}$, $N \in [5, 15]$ and $L = N$ for 1,000 sample points per N. Only RC-RBNs which land in an IS-FSA after being run with inputs of length $T = 10,000$ are used in this test, and RC-RBNs which do not lead to IS-FSA are discarded. Although this test is similar to the IFSA test, we are dealing with a more statistically representative portion of the RC-RBN space, but a less specific subset. This means our results should be taken carefully, as we have no proof which guarantees the trends and properties found in these tests will scale to networks with larger N. With that said, the results of this test can be seen in Fig. 21.

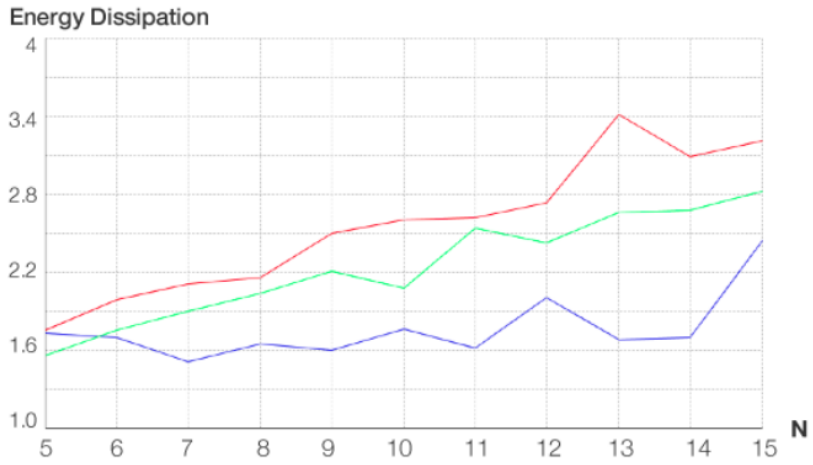

Figure 21: Blue: $<K> = 3$, Red: $<K> = 2$, Green: $<K> = 1$
Using the bound on average energy dissipated per time step as our measure, we extract the IS-FSA from the equivalent FSA for RC-RBNs with the following parameters: $<K> \in \{1, 2, 3\}$, $N \in [5, 15]$ and $L = N$ for 1,000 sample points per N. Only RC-RBNs which land in an IS-FSA after being run with inputs of length $T = 10,000$ are used in this test, and RC-RBNs which do not lead to IS-FSA are discarded.

In interpreting the results of our test shown in Fig. 21 one should realize that as N increases, more networks are thrown away, and the actual sample size is decreased. This requirement of the study decreases the reliability of results as N increases (with constant

sample size) and also accounts for the jagged nature of the plot. Yet, there is a clear trend in that the lower bound on energy dissipated per time step is increasing with N and $<K>$. Once again though, we cannot yet make any claim as to how the parameter, L, affects the energy dissipation.

To study RC-RBNs even further, we now show results of the network's dependence on the parameter, L. Fig. 22 shows a test of a parameter sweep with $<K> \in [1, 3]$, $N \in [5, 10]$ and $L \in [1, 5]$ averaged over 5,000 samples.

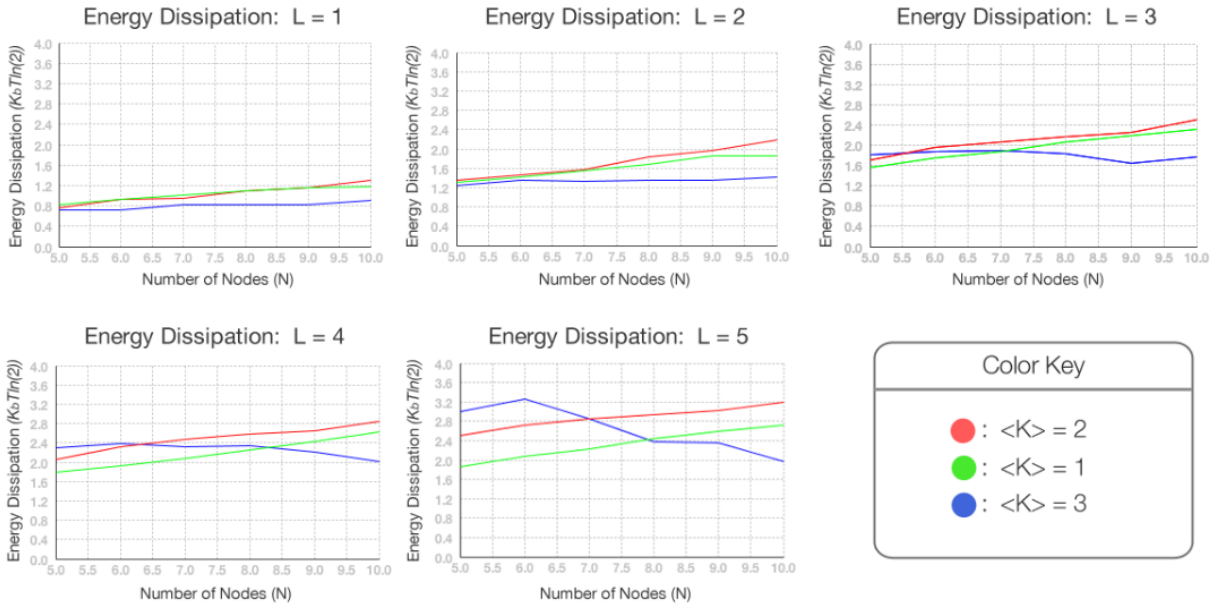

Figure 22: Using the bound on average energy dissipated per time step as our measure, we extract the IS-FSA from the equivalent FSA for RC-RBNs with the following parameters: $<K> \in [1, 3]$, $N \in [5, 10]$ and $L \in [1, 5]$ averaged over 5,000 samples. Only RC-RBNs which land in an IS-FSA after being run with inputs of length $T = 10,000$ are used in this test, and RC-RBNs which do not lead to IS-FSA are discarded. In this test, L is not constrained, and we can see how energy dissipated increases with N and L for $<K> \in [1, 2]$.

Fig. 22 supports the claim that energy dissipation increases with N and also shows that a small change in L can change the dissipation drastically. As L grows, the dissipation bound grows as well, but one interesting observation is how the line corresponding to chaotic

dynamics has a low energy dissipation for *Small-N* networks, but seems to increase in the *Small-N* region for an increasing value of L.

3.5 Discussion

Through this research, we have made a number of interesting findings, and these findings can be split up into three different categories. These are *IFSA*, *IS-FSA* and *General Observations*. The following section synthesizes the results of our study.

IFSA

Our primary goal in this research was to determine if there existed a predictable relationship between energy dissipation and computational capability. SGT shows empirical evidence that computational capability is *maximized* at $<K> = 2$ for all values of T [4]. From our research, and specifically, Fig. 22 we show that for $<K> = 1$ and $<K> = 2$, energy dissipation is an increasing function of both N and L independently. In Fig. 16 we show that there may indeed be a correlation between energy dissipation and computational capability. Together, these three points (maximized computational capability at $<K> = 2$, maximized energy dissipation bound at $<K> = 2$ and correlation between computational capability and the energy dissipation bound for all K) give empirical evidence that energy dissipation is *maximized* at criticality along with computational capability.

With this conclusion in mind, one may ask "What is the the best trade-off?" The trade-off which exists (as per our evidence) is between *short-term memory*, computational capability and energy dissipation in these networks. Specifically, the results of SGT show that a network with ordered dynamics ($<K> = 1$) has a high computational capability when the value of τ is small but quickly attenuates as the value of τ increases (τ a the parameter to computational capability which species the number of similar bits at the end of an input string). Such a response occurs when a system has low short-term memory [4]. In this case, the energy dissipation bound is significantly lower than the case of critical dynamics

($<K> = 2$). Of course, with heterogenous in-degrees, $<K> \in \mathbb{R}$, so the parameter, $<K>$ gives more flexibility in the average in-degree and therefore more flexibility in the system's short-term memory. This increase in control occurs because the value of $<K>$ now *ranges* from 1 to 2 (vs. the case of homogenous in-degree with the parameter K). This trade-off is such that as the value of $<K>$ is lowered, the energy dissipation is lowered, and so is the system's short-term memory (thus lowering computational capability). As the value of $<K>$ increases, the energy dissipation is increased, and so is the system's short-term memory (and thus computational capability). So, if a system were to be built which did not require a long short-term memory, it could be engineered such that the average in-degree be slightly lower than 2 as this would be more energy efficient. Although the network's in-degree gives one trade-off, the number of inputs to the network give another. Specifically, the higher the value of L, the higher the energy dissipation and the larger the system's short-term memory.

As it stands, we have provided evidence of a relationship between computational capability and energy dissipation in *Small-N* RC-RBNs. However, we have no proof to claim that this relationship will hold as the number of network nodes increases. Because of this, more research needs to be done to show this is true for N>15. Intuitively speaking though, the simple linear relationship will probably not hold as the number of nodes increases, because depending on the number of input connections, computational capability resembles a logarithmic function in the ordered and critical phases and a decreasing exponential in the chaotic phase.

IS-FSA

Aside form our primary result, we have shown interesting relationships between RC-RBNs and their equivalent FSA. We have shown that the number of IFSA mapping RC-RBNs decreases exponentially with increasing N. This is interesting, because the same can be said for the percentage of state nodes involved in IS-FSA. Intuitively it makes sense though, because IS-FSA are a generalization of IFSA (IFSA are IS-FSA which include every state in

the FSA). We also showed that IS-FSA have the same phase ordering as IFSA, which is also to be expected. We showed that the size of IS-FSA increase with N, L and $<K>$ independently (for the lower half of *Small-N*) networks, but that as N increases, the phase ordering of this measure match closely with that of computational capability for similar parameters.

General Observations

Throughout most of our studies, there is an obvious trend in the way the value of the $<K> = 3$ plot lines tend to start high and drop as N increases. What is most interesting is that this trend occurs in both the computational capability test (a study on the wiring diagram) and in the other tests (studies of the state network). All of these show the descending trend in the values of $<K> = 3$ and show that the other values of $<K>$ tend to keep their same ordering. Although we do not yet have a definite explanation for this phenomenon, one may realize that even a chaotic network may act in a useful way with a limited number of nodes.

4 Conclusions and Future Implications

In this work, an open source program was written for the simulation of reservoir computer models. These models used random Boolean networks as their internal reservoirs, and the simulator was given a number of unique features. Two critical features being the ability to convert an arbitrary reservoir computing model with a random Boolean network reservoir to an equivalent finite-state automata and the ability to inspect any irreducible finite-state automata for its steady state energy dissipation bound. In the making of this simulator, we also developed a number of algorithms to aid in our experiments. Specifically, we developed algorithms to convert arbitrary reservoir computing models with random Boolean network reservoirs into equivalent finite state automata, test for finite state automata irreducibil-

ity, extract irreducible-sub finite state automata, and perform the summations required to quantize the energy bound given in [10].

Our research objectives were motivated by the following set of general questions:

1. How can we obtain a minimum bound on the average energy dissipated per time step for any instance of an RC-RBN, and how is this related to computational capability?

2. How do the parameters of an RC-RBN affect its equivalent FSA, and how do properties of the FSA relate to the dynamics of the original RC-RBN?

3. How does the phase of an RBN (ordered, critical, chaotic) affect the minimum bound on the average energy dissipated per time step for that RC-RBN?

In terms of these general research goals, we were able to both obtain and show a minimum bound on the average energy dissipated per time step for an instance of a reservoir computer using a random Boolean network as its internal reservoir. We were also successful in showing how this bound related to computational capability, but we were restricted to networks with fewer than 15 nodes. We were successful in showing how parameters in the reservoir computer affected its equivalent finite-state automata, but many of these phenomena have not yet been explained. We were also able to show that the phase of a random Boolean network acting as a reservoir has a significant impact on the energy dissipation bound. Specifically, energy dissipation is highest when the average in-degree to each node is 2.

Although these are the questions which motivated our objectives, our overall motivation for this work comes from the void in literature which exists in the use of reservoir models that are capable of heterogenous in-degrees. Our primary motivation came from the work of SGT [4] where reservoir computers are analyzed with random Boolean networks for the first time and applied to the motivation of creating self-assembling and physical devices [4].

As we developed our research methodology, the biggest issue we had was handling the size of the equivalent FSA. As the number of nodes in the RBN grew, the number of states in the FSA grew exponentially up to the point where the program ran out of memory. We

also faced some issues with applying the bound presented in [10] to our work. We learned early on that RC devices with RBN reservoirs do not map well to irreducible FSA (see Sec. 3.1). So, we were forced to alter this bound for our own use by introducing the concept of irreducible-sub FSA. Before this, it was unfeasible to study networks with any more than 7 nodes. After adopting the bound, we were able to analyze networks with up to 15 nodes in a "reasonable" amount of time. With both of these issues, we decided to study RBNs with fewer than 15 nodes and use small sample sizes to counter the longevity of running our depth-first search algorithms.

Through our research, we have shown empirical evidence of correlation between the measure of computational capability [21] and the lower bound on average energy dissipated per time step [10] for reservoir computing devices using random Boolean networks with fewer than 15 nodes. We developed the concept of an irreducible-sub finite state automata and developed algorithms to pull them from finite state automata. We were able to show that the probability of generating irreducible-sub finite state automata decreases exponentially with an increase in the number of nodes in the generated random Boolean networks. We have also shown that the size of irreducible-sub finite-state automata tend to increases linearly and independently for the number of nodes in the network, the number of input signals and the node in-degree for random Boolean networks with fewer than 10 nodes. We also noticed a general trend in the behavior of values corresponding to chaotic dynamics, specifically, the values start high and consistently go low as the number of nodes in the network increase.

Although we presented these results for networks with fewer than 15 nodes, future work needs to be done in performing similar tests for more realistic network sizes. If there is a predictable relationship between the computational capability of these networks and the energy dissipation, then engineers may be able to use this knowledge to their advantage in creating physical devices which can be modeled as reservoir computers with heterogenous reservoirs.

References

[1] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[2] M. Lukoševičius, H. Jaeger, and B. Schrauwen, "Reservoir computing trends," *KI-Künstliche Intelligenz*, vol. 26, pp. 365–371, 2012.

[3] F. Duport, B. Schneider, A. Smerieri, M. Haelterman, and S. Massar, "All-optical reservoir computing," *Optics Express*, vol. 20, pp. 22783–22795, 2012.

[4] D. Snyder, A. Goudarzi, and C. Teuscher, "Computational capabilities of random automata networks for reservoir computing," *Physical Review E*, vol. 87, p. 042808, 2013.

[5] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, pp. 183–191, 1961.

[6] J. Von Neumann, A. W. Burks, *et al.*, *Theory of self-reproducing automata*. Illinois: University of Illinois press Urbana, 1966.

[7] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of theoretical biology*, vol. 22, pp. 437–467, 1969.

[8] C. Gershenson, "Introduction to Random Boolean Networks," *eprint arXiv:nlin/0408006*, 2004.

[9] H. A. Carteret, K. J. Rose, and S. A. Kauffman, "Maximum power efficiency and criticality in random boolean networks," *Phys. Rev. Lett.*, vol. 101, p. 218702, 2008.

[10] N. Ganesh and N. G. Anderson, "Irreversibility and dissipation in finite-state automata," *Physics Letters A*, vol. 377, pp. 3266–3271, 2013.

[11] A. Wuensche, "Discrete dynamical networks and their attractor basins," 1998.

[12] S. A. Kauffman, *Investigations*. Oxford University Press, 2000.

[13] B. Luque and R. V. Solé, "Phase transitions in random networks: simple analytic determination of critical points," *Physical Review E*, vol. 55, pp. 257–260, 1997.

[14] B. Derrida and Y. Pomeau, "Random networks of automata: A simple annealed approximation," *EPL (Europhysics Letters)*, vol. 1, p. 45, 1986.

[15] I. Harvey and T. Bossomaier, *Time Out of Joint: Attractors in Asynchronous Random Boolean Networks*. MIT Press, 1997.

[16] H. K.A, J. H.A, and S. C.J, "Simulating large random boolean networks," *Res. Lett. Inf. Math. Sci.*, vol. 11, pp. 33–43, 2007.

[17] C. H. Bennett, "The thermodynamics of computationa review," *International Journal of Theoretical Physics*, vol. 21, pp. 905–940, 1982.

[18] A. Bejan, *Entropy generation minimization: The new thermodynamics of finite-size device and finite-time processes*. CRC Press, 1996.

[19] I. Shmulevich and S. A. Kauffman, "Activities and sensitivities in boolean network models," *Physical Review Letters*, 2004.

[20] I. Shmulevich, E. R. Dougherty, S. Kim, and W. Zhang, "Probabilistic boolean networks: a rule-based uncertainty model for gene regulatory networks," *Bioinformatics*, vol. 18, pp. 261–274, 2002.

[21] T. Natschläger and W. Maass, *Advances in Neural Information Processing Systems*, vol. 17. Cambridge: MIT Press, 2004.

[22] H. F. Nelson, "Random readme." Unpublished Notes, 2004.

5 Appendix

In this section you will find several algorithms we have used in developing our simulations. Alg.1 takes an irreducible finite-state automata as an input, and outputs the minimum bound on average energy dissipated per time step as given in [10]. Alg. 2 presents the code used to convert a reservoir computer / random Boolean network to an equivalent finite-state automata. Alg. 4 and 3 give the pseudocode to determine whether or not an instance of a finite-state automata is irreducible. The first gives an algorithmic approach and the second gives a matrix decomposition approach. Alg. 5 gives an algorithmic approach to extracting irreducible-sub finite-state automata from a general finite-state automata.

```
Data: ifsa = An Instance of an IFSA, q = input pmf
Result: Minimum average energy dissipated per time step
π = Steady state occupation vector
informationLoss = 0
foreach Input j ∈ Inputs do
    stateSum = 0
    foreach State s in States do
        entropiesOfStatesWhichLeadToS = 0
        foreach State L in States which lead to s do
            pL = Probability the previous state was L given that the current state is S
            given input j (using π)
            eL = Entropy of pL
            entropiesOfStatesWhichLeadtoi += eL
        end
        stateSum += entropiesOfStatesWhichLeadtoi
    end
    informationLoss += (sumOfStates)(q_j)
end
return (informationLess)(-1)
```

Algorithm 1: IFSA minimum bound on energy calculation.

Data: rbn = An Instance of an RBN

Result: fsa = An Instance of an FSA

N = Number of nodes in the rbn

RBN_States = Number of RBN States = 2^N

fsm = new FSM with S States and One input

foreach *State s in RBN_States* **do**

reinitialize rbn to state s

current_Output = get the current output of rbn

move rbn foreword one time step with an input of 0

zero_nextState = the current state of rbn

reinitialize rbn to state s

move rbn foreword one time step with an input of 1

one_nextState = the current state of rbn

set state s of fsa to goto state zero_nextState when given an input of 0

set state s of fsa to goto state one_nextState when given an input of 1

set the output of state s of fsa to current_Output

end

return fsm

Algorithm 2: Converting an RBN to an FSA

List of IS_FSA `getAll_IS_FSA_ViaMatrix()`

```
/* Get the Eigenvectors                                    */
```

matList = getEigenVectorsForEigenValueOne()

return matList

Algorithm 3: Matrix Method to Find IS-FSA

```
Boolean isIrreducible(FSA)
    M = Number of states in FSA
    Matrix P = new Empty MxM Matrix
    foreach State S in FSA do
        foreach State (transition) S can lead to do
            P[S, transition] = 1 updateState_S0_ThroughState_S1(P, S, transition)
            stateTransitionDidChange(P, S)
        end
        if P has all ones then
            return true
        else
            return false
        end
    end
void stateTransitionDidChange(TransitionMatrix P, State changedState)
    foreach State S in P do
        if S is connected to changedState then
            updateState_S0_ThroughState_S1(TransitionMatrix P, State S, State
            changedState)
        end
    end
void updateState_S0_ThroughState_S1(TransitionMatrix P, State S0, State S1)
    bool hasSChanged = false foreach State U in S which is not connected but is
    connected in S1 do
        bool hasSChanged = true
        foreach S is connected to changedState do
            P[S0, U] = 1
            hasSChanged = true
        end
        if hasChanged then
            stateTransitionDidChange(P, S)
        end
    end
```

Algorithm 4: Algorithmic Irreducibility Test

```
IS_FSA getIS_FSA_ViaAlgorithm(currentState)
    /* Create blank lists                                              */
    stateChain = Create a new blank list of Integers solutionChains = Create a new
    blank list of lists of Integers failStates = Create a new blank list of Integers
    successStates = Create a new blank list of Integers
    /* Find chains of states which lead from the starting state and lead
       back to the starting state and put them in ''solutionChains''   */
    depthFirst_FindSolutionChains(currentState, currentState, stateChain,
       solutionChains, failStates, successStates) /* Determine which states can
       be reached within this IS-FSA                                   */
    solutionStates = Create a new blank list of Integers foreach solution in
    solutionChains do
        foreach state in solution do
            if solutionStates does not contain state then
                Add state to solutionStates
            end
        end
    end
    if there are no solution then
        return null
    end
    /* Determine if these solution states are actually an IS=FSA       */
    foreach state in solutionStates do
        if state leads to a state not in the set of solution states then
            return null
        end
    end
    /* Now that we have an IS-FSA we can create a new IFSA             */
    fsa = Create a new FSA the size of solutionStates map = Create a mapping from
    old state numbers to new state numbers return a new IS-FSA with fem and map
```

Algorithm 5: Algorithmic Method to Find IS-FSA