Reservoir Computing with Random DNA Strand Displacement Circuit Systems

Hoang Nguyen[1], Peter Banda[2], Darko Stefanovic[3], and Christof Teuscher[1]

[1] Portland State University, Portland OR 97207, USA
{hoang24,teuscher}@pdx.edu
http://www.teuscher-lab.com
[2] University of Luxembourg
peter.banda@uni.lu
https://peterbanda.net
[3] University of New Mexico
darko@cs.unm.edu
http://www.cs.unm.edu/~darko

Abstract. Top-down engineering of biomolecular circuits to perform specific computational tasks is notoriously hard and time-consuming. Current circuits have limited complexity, are brittle, and application-specific. Here we propose an alternative; we design and test a bottom-up constructed *Reservoir Computer* (RC) that uses random DNA strand displacement circuits. This RC has the potential to be implemented easily and trained for various tasks. We describe and simulate it by means of a *Chemical Reaction Network* (CRN) and we evaluate its performance on three computational tasks: the Hamming distance and a short- as well as a long-term memory task. Compared with the deoxyribozyme oscillator RC model simulated by Yahiro *et al.*, our random DNA RC performs 68.2% (in NRMSE) better for the short-term and 60.8% (in NRMSE) better for the long-term memory task; however, our model relies on a 82% larger variety of chemical species. Our model successfully solved the Hamming distance between two input bitstreams, with the average NRMSE of 0.18 ± 0.38.

Keywords: Random DNA strand displacement circuit · Reservoir computing · Hamming distance learning · Short-term memory task · Long-term memory task

1 Introduction

Implementing a top-down chemical system relies on reasoning about the functioning of the system parts in sequential causal pathways, well isolated from each other. This rather conservative approach tries to avoid complications arising from non-linear dynamics, inherent parallelism, and concurrency of chemistry, and therefore considers them adversary. Current molecular machines such as reprogrammable DNA self-assembly [25] requires the sets of molecules and reactions to be explicitly designed. Chemical systems whose functionalities can be reprogrammed or adjusted by autonomous learning have been explored using abstract CRNs [15–17], an enzymatic chemistry [18], as well as buffered DNA strand displacement circuits [20,21]. We argue that a bottom-up approach, where the species and reactions of chemical systems are selected at random, could explore the functional landscape, its phase transitions, and dynamical regimes beyond intuition by embracing these properties.

A DNA strand displacement circuit [4,26] is undeniably the most popular choice for a bottom-up constructed chemistry. This is due to Soloveichik's proof [4] of a universal approximation of mass-action driven CRNs, which showed that complex CRNs can be programmed using DNA-based chemistry. Random chemical networks have been investigated primarily in the origins of life literature [12,27], where several kinds of mostly (auto)catalytic reactions occurring in primordial soup throughout the evolution provided the basis for closure and homeostasis. Experimentally, a network of peptides assembled randomly exhibited self-organization for predicted network connectivity [28]. Furthermore, random catalytic networks produced self-replication [13] and oscillation [14]. Nevertheless, compared with non-chemical circuits, such as neural and Boolean networks, randomness in chemistry remains relatively unexplored.

In this paper, we propose that the dynamics of a random DNA strand displacement circuit system is an ideal candidate for reservoir computing, an emerging computing architecture. A *Reservoir Computer* (RC) [7,29] consists of a fixed, randomly connected recurrent neural network, the reservoir, which acts as a set of high-dimensional filters with fading memory, and a memoryless readout layer, trained by supervised learning. Reservoir computing has a fairly simple mathematical model and can outperform standard machine learning algorithms especially for temporal (time series) tasks. Without relying on specific species/reaction design or initial concentration, the random DNA strand displacement circuit can achieve complex dynamics that translates to superior learning performance. This provides inherent non-linearity and several types of dynamical regimes that are useful for designing and building an RC.

In DNA reservoir computing, Goudarzi *et al.* and Yahiro *et al.* have successfully used deoxyribozyme oscillators [8,9] to solve temporal problems [2,3]. Here we show that a random DNA strand displacement circuit, an alternative chemical building block for reservoir computing, can achieve better performance in solving temporal tasks than the deoxyribozyme oscillator RC. In particular, the random DNA strand circuit RC achieves a 68.2% and 60.8% improvement in performance (NRMSE) compared with the deoxyribozyme oscillator RC in Yahiro *et al.* This RC also successfully learns the Hamming distance between two input bitstreams, with the average NRMSE of 0.18 ± 0.38 over four different input hold times. These tasks demonstrate the performance and feasibility of the random chemical RC and pave the way for potential wet applications such as detecting pathogens or gene mutations.

2 Approach

In this section, we discuss how to generate the network of a random DNA strand circuit and to achieve the most dynamical behavior using optimization methods. Then, we describe how to apply the Gillespie algorithm [1] as a means to run our CRN model. At the end of the section, we show an example of a random DNA strand circuit setup and simulation using the optimized parameters.

2.1 Random DNA Strand Circuit Species and Reactions

A random DNA strand circuit network is a Chemical Reaction Network (CRN) [6], which consists of a set of species and reactions [5]. There are four types of species in the network: upper strand, lower strand, partial double strand, and full double strand. Upper and lower strands are single-stranded DNA molecules such that only opposite strand types can bind. We assume this property is guaranteed by sequence design [11]. What we refer to as full double strand is a perfect Watson-Crick DNA double strand, where all base pairs of corresponding upper and lower strands are complementary. Partial double strand is similar to full double strand but instead of all pairs, only substrings of upper and lower strands match.

These DNA strands react with one another under four types of reactions: binding, displacement, influx, and efflux. Only upper single strands and lower single strands react in binding reactions. The result of a binding reaction can be a full double or a partial double strand. A displacement reaction happens between a single strand and a partial double strand. In this type of reaction, the single strand displaces the partial double strand into two single strands, and binds to one of these strands to become a full double or a partial double strand.

There are two important notes about strand displacement [4]: 1) Full double strands cannot be displaced by single strands and are considered the strongest, 2) Only stronger strands can displace weaker strands and not the other way around. The strength of a strand is defined as the length of the toehold in the strand. Fig. 1 illustrates different variations of strand binding and displacement reactions.

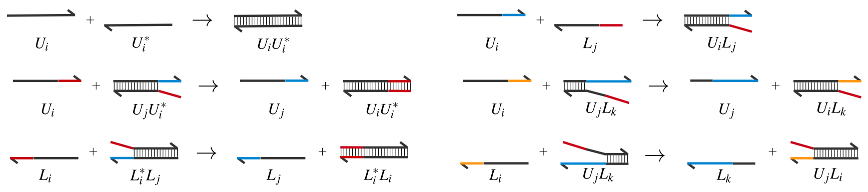

Fig. 1. Assumed DNA strands binding and displacement reactions (left-to-right, top-to-bottom order): a) full double strand creation, b) partial double strand creation, c) upper strand displaced by full complement, and d) upper strand displaced by another hierarchically stronger strand, e) f) lower strand versions of full and partial strand displacements. Upper strands are labeled as U, lower as L. The star ($*$) notation indicates complementarity.

Using these DNA computing principles as a building block, we can set up and test our random DNA strand circuit model. First, we select a 10 μL reservoir volume to work with, since we assume the system is a microscale continuous stirred-tank reactor (μCSTR) [8, 9], which has small volume for achieving high species concentration. The volume will be used to determine the influx and efflux reaction rates of species inside the chamber.

2.2 Random DNA Strand Circuit Generation

We generate the network of a random DNA strand circuit using nine input parameters. There are five general parameters: the number of single strands n, the ratio of upper to lower strands ρ, the ratio of upper strands with complements γ, the ratio of influx to the overall number of strands α_{in}, and the ratio of efflux to the overall number of strands α_{out}. There are four random distribution parameters: the normal distribution of rate constant θ, the normal distribution of influx rate θ_{in}, the normal distribution of efflux rate θ_{out}, and the normal distribution of partial double strands per upper strand ϕ.

We then apply the network generation process to generate the random DNA strand circuit network. Specifically, these steps determine the types and the names of DNA strands in the network, the order of strands for strand displacement reactions, and the influx/efflux of the chemistry, with respect to the DNA network parameters described above.

```
Input: Parameters $n$, $\rho$, $\gamma$, $\phi$, $\theta$, $\alpha_{in}$, $\alpha_{out}$, $\theta_{in}$, $\theta_{in}$
Output: DNA Strand Network
Determine $n_L = n \div (1+\rho)$, and $n_U = n - n_L$.
Create upper and lower strands as nodes of the network.
Determine $n_F = \gamma \times min(n_L, n_U)$.
Choose $n_F$ complementary upper and lower DNA strands randomly. Connect
  them with solid lines.
for each upper DNA strand do
    Draw $n_P$ partial double strands from the distribution $\phi$.
    Choose randomly given $n_L$ counterparts without repetitions.
    Omit lower strand from selection if already selected as complementary.
end
Mirror selection of partial double strands for upper strands with complements in
  the pool.
Impose ordering of partial double strands for strand displacement reactions so
  only DNA strands with higher order displace strands from the complex.
Choose random influx and efflux strands for parameters $\alpha_{in}$ and $\alpha_{out}$.
Generate random rate constants from the distribution $\theta$.
Generate random rate constants for influxes and effluxes from the distribution $\theta_{in}$
  and $\theta_{out}$.
```

Algorithm 1: Random DNA strand circuit network generation process. n_L, n_U, n_F, n_P refer to the number of lower strands, the number of upper strands, the number of full double strands, and the number of partial double strands.

The ordering corresponds to the length of the overlapping sub-sequences for a given upper or lower strand so only the strands with longer overlap kick out those with shorter overlap from the complex. Note that fully complementary Watson-Crick strands always have the highest order over all competing strands. Now, we impose the ordering of partial double strands globally, where the partial double strands are ordered from the perspective of the entire system. That prevents cyclic displacement and is easier to construct. Global ordering uses essentially the same trick employed in operating systems to prevent live-lock of resources.

2.3 Parameters Optimization

The random DNA strand circuit parameters were selected using the *Turning Point Measure* (TPM) experiments and a *Genetic Algorithm* (GA). A TPM is a new measure that we propose to express non-monotonicity of system dynamics. This method quantifies the number of times the system changes direction in a given time period. A Turning Point (TP) is a local minimum or maximum of the function, a stationary point where the derivative (gradient) changes sign from + to - and vice versa. More formally, x is a TP of the *i*-th variable of continuous-time dynamical system driven by $f : \mathbb{R}^n \to \mathbb{R}^n$, denoted as a predicate TP_i if

$$TP_i(x) \Leftrightarrow \frac{\partial f}{\partial x_i}(x) = 0 \text{ and } \frac{\partial f}{\partial x_i}(x_{-\varepsilon}) = -\frac{\partial f}{\partial x_i}(x_{+\varepsilon}) \tag{1}$$

We then define the *Turning Point Count* (TPC) of the function f at interval $(a,b) \in \mathbb{R}^n \times \mathbb{R}^n$:

$$TPC(f) = |\{x \in (a,b) \mid \exists i \in \{1,...,n\}\colon TP_i(x)\}| \tag{2}$$

By normalizing the TPC in time and space, we obtain the TPM:

$$TPM(f) = \frac{TPC(f)}{n(b-a)} \tag{3}$$

In the genetic search, a standard GA was employed. The vector of network generation parameters is encoded in a chromosome of possible solutions. The fitness is the average TPM over 1,000 randomly generated DNA strand circuits using the network generation parameters in the chromosome. We used 10,000 time steps, and generated random initial concentrations on the (0,1) interval.

From the TPM and the genetic search, these parameters were optimized and can be used to generate the random DNA strand circuit with the most interesting dynamics. Table 1 gives the optimized parameters and their values that we used to characterize our CRN. In the experiments that follow, we used these parameter values to generate the network.

2.4 Simulating the Chemistry

We use the Gillespie algorithm [1] to simulate the random DNA strand circuit. There are four steps in this algorithm: initialization, Monte Carlo step, update step, and iterate.

Initialization: The initial number of molecules is set at random for each DNA strands, since we want a fairly large population in order to achieve interesting dynamics. After the chemistry starts running, the entire system remains unperturbed for an amount of time t_p. At time t_p, we start introducing perturbations to the chemistry. The amount of time between two consecutive perturbations is the hold time τ and the maximum simulation time is t_{max}. The number of perturbations N is:

$$N = \frac{t_{max} - t_p}{\tau} \tag{4}$$

Table 1. Values of DNA strand circuit network generation parameters discovered after the TPM and genetic search.

General Parameter	Value
n	7
ρ	0.75
γ	1/3
α_{in}	0.99
α_{out}	0.10
Normal Distribution Parameter	Value
θ	0.075 ± 0.01
θ_{in}	0.0003
θ_{out}	0.0003
ϕ	3 ± 0.001

Monte Carlo step: Reaction rates for four types of reactions are generated as follow:

For displacement and binding reactions:

$$r_i = k_i[X_1][X_2] \tag{5}$$

For efflux reactions:

$$r_i = \frac{k_i[X]}{V} \tag{6}$$

For influx reactions:

$$r_i = \frac{k_i}{V} \tag{7}$$

Let n_R be the number of reactions. The total reaction rate is:

$$r_{tot} = \sum_{i=1}^{n_R} r_i \tag{8}$$

The probability for a reaction to happen is determined by:

$$P_i = \frac{r_i}{r_{tot}} \tag{9}$$

The time step in the Gillespie algorithm is exponentially distributed with a mean of $1/r_{tot}$.

Update step: After a reaction happens, we update the species concentration and the simulation time. The time we run the next iteration in the Gillespie algorithm is determined by:

$$t_{i+1} = t_i + dt, \tag{10}$$

with dt being the random number drawn from the exponential distribution with mean $1/r_{tot}$.

Iterate: Repeat Monte Carlo and Update steps. The stopping conditions are: (1) The maximum simulation time is reached ($t \geq t_m$) or (2) The molecular count drops to 0 ($[X] = 0$).

Sample run: Below is a sample setup and simulation of the random DNA strand circuit using the optimized network parameters (see Fig. 2 for the setup and Fig. 3 for the concentration plot.)

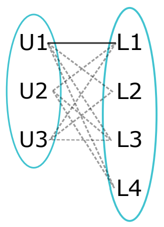

Fig. 2. Example network representation of DNA strand displacement reactions. Nodes are single strands, which divide the network into upper strands U (left), and lower strands L (right). Bold lines represent full double strands, dashed lines represent partial strands. For this setup, the species are: Single (U1, U2, U3, L1, L2, L3, L4); Full double (U1L1); Partial double (U2L1, U1L3, U1L4, U2L1, U2L3, U2L4, U3L1, U3L2, U3L3).

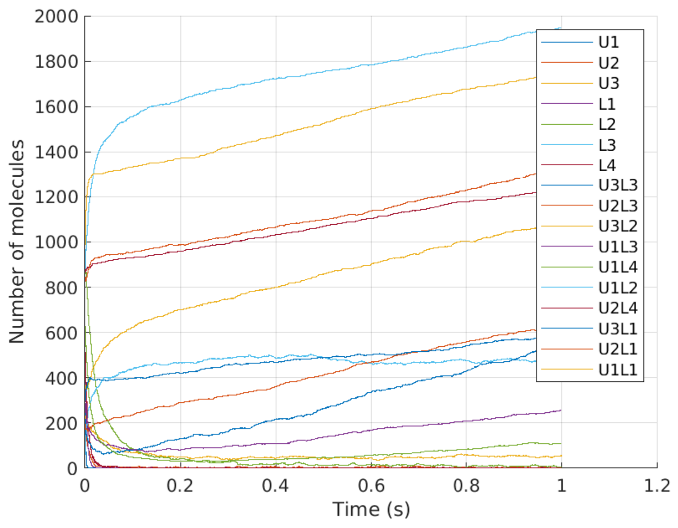

Fig. 3. Sample concentration plot showing the behaviors of the species within the random DNA strand circuit, with the total simulation time of 1 second.

3 Applying Reservoir Computing to Random DNA Strand Circuit

3.1 Reservoir Computing Overview

Reservoir Computing is a promising computing technique that provides real-time computing on reservoir transitions without the need of stable states. RCs outperform the classic recurrent neural networks on time series processing and prediction. The core of an RC (the reservoir/liquid) could be generated at random, while only the memoryless readout layer is used for training [7]. A reservoir provides both a fading memory of the past inputs through its inherent recurrency and computing power due to its non-linear dynamics (high dimensional filters) operating at the regime that promotes balanced information transfer.

The best known instances of RC are echo state network [30, 31] and liquid state machine [32]. We argue that the basic idea of RC in its most general form—process inputs through a randomly generated component with rich dynamics and then map its states to the output layer by means of simple linear integration—also applies to domains beyond the neural-network or logic circuit formalisms. Most of its implementations are recurrent neural networks, however, models of liquid state machine (an instance of reservoir computing) in a bucket of water [33] and in E. Coli [34] demonstrate that this approach could extend spatial or network based systems.

A simple RC consists of three components: the inputs of the reservoir, the main reservoir, and the readout layer giving the outputs of the reservoir. Sketches of RCs are shown in Fig. 4 and Fig. 5.

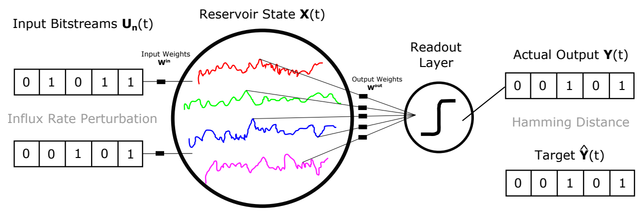
Input Bitstreams $\mathbf{U_n}(t)$
Reservoir State $\mathbf{X}(t)$
Input Weights $\mathbf{W}^{in}$
0 1 0 1 1
Influx Rate Perturbation
0 0 1 0 1
Output Weights $\mathbf{W}^{out}$
Readout Layer
Actual Output $\mathbf{Y}(t)$
0 0 1 0 1
Hamming Distance
Target $\hat{\mathbf{Y}}(t)$
0 0 1 0 1

Fig. 4. Abstract RC for learning the Hamming distance between two input bitstreams.

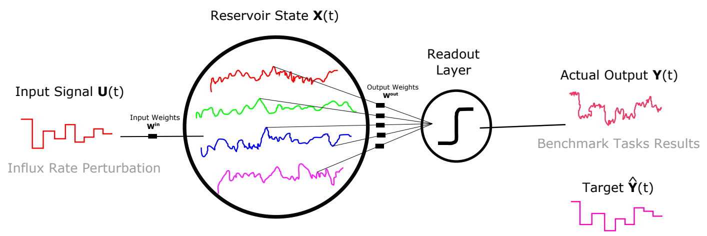
Reservoir State $\mathbf{X}(t)$
Input Signal $\mathbf{U}(t)$
Input Weights $\mathbf{W}^{in}$
Influx Rate Perturbation
Output Weights $\mathbf{W}^{out}$
Readout Layer
Actual Output $\mathbf{Y}(t)$
Benchmark Tasks Results
Target $\hat{\mathbf{Y}}(t)$

Fig. 5. Abstract RC for learning the short- and long-term memory task.

In a nutshell, RC accepts time-series inputs, which change the dynamics inside the main reservoir. The readout layer then reads the information inside the reservoir and uses it for a specific task.

Let I be the number of reservoir inputs, N the number of nodes inside the reservoir, and O the number of reservoir outputs. Let i the input node index, j and k the reservoir node indexes, and l the output node index. For the input side, we introduce $U(t) = [u_i(t)]$ as the I-dimensional input vector, and $W^{in} = [w^{in}_{ij}]$ as the $N \times I$ input weights matrix. For the main reservoir, we introduce $X(t) = [x_j(t)]$ as the N-dimensional vector of the reservoir state, and $W^{res} = [w^{res}_{jk}]$ as the $N \times N$ reservoir weights matrix. Lastly, on the readout side, we introduce $Y(t) = [y_k(t)]$ as the O-dimensional output vector, and $W^{out} = [w^{out}_{kl}]$ as the $O \times N$ output weight matrix.

The reservoir state vector $X(t)$ can be determined as:

$$X(t+1) = f(W^{res} \cdot X(t) + W^{in} \cdot U(t)) \tag{11}$$

Here f is the non-linear transfer function of the reservoir nodes. The output vector $Y(t)$ can also be determined using linear combination of the $X(t)$ vector,

$$Y(t) = W^{out} \cdot X(t) + w_b, \tag{12}$$

where w_b is an inductive bias. Since the purpose of reservoir computing is only training the output weights, the input weights matrix W_{in} and reservoir weights matrix W_{res} are kept fixed. The output weights matrix W_{out} can be trained using any linear regression method. Similar to [2] and [3], we employed the simple Moore-Penrose pseudo-inverse method [10]:

$$W^{out'} = (X'^{T} \cdot X')^{-1} \cdot X'^{T} \cdot \hat{Y}' \tag{13}$$

In this equation, $W^{out'}$ is the matrix W^{out} with the bias w_b added. X' is the observation matrix of the reservoir, where each row represents the reservoir state in time and each column represents the state at different nodes such that the last column is a constant 1. Lastly, $\hat{Y}'$ is the target vector.

3.2 Using Random DNA Strand Circuit for Reservoir Computing

In our random DNA strand circuit system, the dynamics of the chemistry is characterized by DNA strands species. This system of random DNA strand circuit can be used as a RC, with the influx rates being the reservoir input and the chemical dynamics of the DNA strands being the blackbox signals inside the reservoir. The readout/output layer of the RC can be trained to learn different machine learning tasks. Fig. 6 below shows an abstract reservoir computing unit made of random DNA strand circuits.

The species inside the reservoir can be perturbed by changing their influx rates. This changes the concentrations of the influx species and the concentrations of the other species that react with these influx species. The number of reservoir inputs depends on the number of species being perturbed in the random DNA strand circuit. For instance, if we design a two-input reservoir, then we need to perturb two DNA species while running the chemistry.

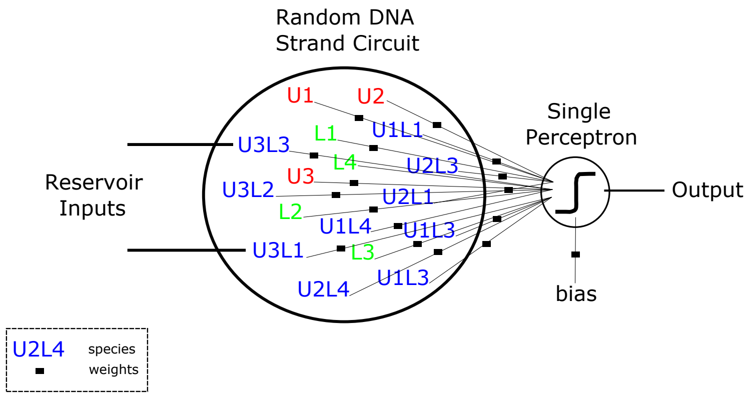

Fig. 6. RC model using random DNA strand circuit.

The main reservoir consists of the information about the types and concentrations of all the DNA species present in the random DNA strand circuit at different times during the chemistry simulation. The setup includes 17 DNA species, so the main reservoir contains the concentration dataset of these species. The species concentration information inside the main reservoir is then weighted with uniformly distributed random number, and is projected into a readout layer consisting of a single perceptron. This perceptron uses a feedforward and backpropagation technique, employing a simple sigmoid function to fit the computed output into a desired target using weighted information. This happens through a number of learning iterations, with a set learning rate.

Depending on the purpose of each task, we can determine the target, inputs, and readout layer function for the reservoir to learn the tasks. Other perceptron parameters such as the bias, the learning rate, and the number of iterations need to be adjusted as needed to help increase the performance of the readout layer.

We use the NRMSE to evaluate the performance of our RC model and compare with other DNA RC models. The error is defined as:

$$NRMSE = \frac{1}{Y_{max} - Y_{min}} \sqrt{\frac{\sum_{t=t_1}^{t_n} (Y(t) - \hat{Y}(t))^2}{n}} \tag{14}$$

Here, n is the length of the time vector.

3.3 Hamming Distance Learning

The first problem we chose to tackle with the random chemical reservoir is the Hamming distance between two bitstreams. The goal is to learn the Hamming distance between two input bitstreams. These inputs are fed into the reservoir and the number of bits of an input is defined as the number of times the chemistry is perturbed.

Methodology We chose two species to be the influx species. After a defined input hold time τ, we changed the influx rates of these species, which perturbed the system. The vector of influx rates were then converted into two bitstreams. The target of the task is the expected Hamming distance between those two bitstreams.

Next, we construct the readout layer with the information of species concentration. We can feed this information into the perceptron and start the training as described in Section 3.2. Last but not least, we need to check if the output of the perceptron agrees with the target vector and check if the learned Hamming distance agrees with the calculated Hamming distance.

After running multiple experiments with different setups for the random DNA strand circuit, we observed that the reservoir computer was able to correctly learn the Hamming distance with small learning error.

Here, we introduce four neural network parameters: the number of iterations (number of times that the readout layer is trained), the learning rate (how fast the readout layer can learn a task), the number of bits (number of bits in the input, output, and target vectors), and the bias to help the output fit better to the target.

Results Fig. 7 shows the NRMSE of the learned Hamming distance with respect to the input hold time. Each setup with a different input hold time is averaged over 50 simulations. The average NRMSE over four different input hold times is 0.18 ± 0.38.

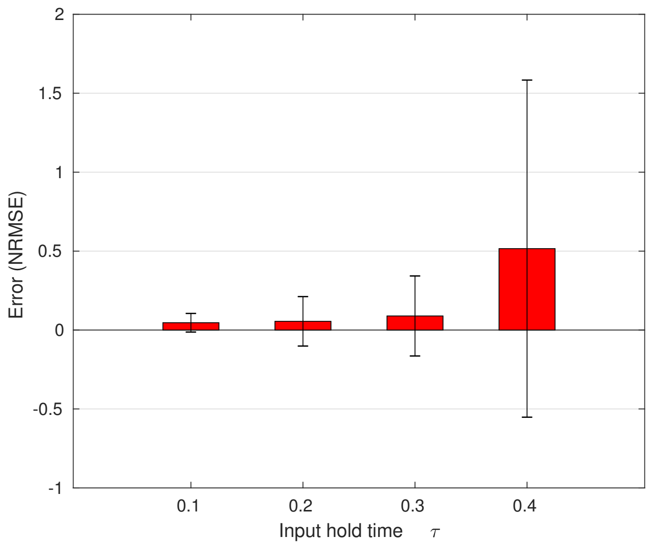

Fig. 7. NRMSE of the Hamming distance between two input bitstreams with various lengths, determining by the input hold time τ, learned by the random DNA strand circuit RC. The base influx rate is kept fixed at 0.0003, while the input hold time τ is swept from 0.1 to 0.4 with a 0.1 increment. As the input hold time increases, the error gets higher. The whiskers represent the standard deviation over 50 simulations. This can be reduced by averaging over a larger number of simulations.

3.4 Temporal Tasks Learning

We test the random DNA strand circuit RC with two temporal learning tasks: the short-term memory task and the long-term memory task. The goal of the tasks is to train the readout layer so that its output over a number of iterations is close to the target. The NRMSE introduced above is used as a means to measure the performance of the RC.

Short-term memory task The short-term memory task is defined as:

$$\hat{Y}(t) = S_k^m(t-1) + 2S_k^m(t-2) \tag{15}$$

$\hat{Y}(t)$ is the target vector as a function of time t. $S_k^m(t)$ is the influx rate $S^m(t)$ of the kth species, as a function of time.

Long-term memory task The long-term memory task is defined as:

$$\hat{Y}(t) = S_k^m(t-\tau) + \frac{1}{2}S_k^m(t-\frac{3}{2}\tau) \tag{16}$$

$\hat{Y}(t)$ is the target vector as a function of time t. $S_k^m(t)$ is the influx rate $S^m(t)$ of the kth species, as a function of time. τ is the time between each perturbations (the input hold time).

Results We compute the NRMSE for the short-term and long-term memory task as a way to measure the performance of the random DNA strand circuit RC, and to compare with the deoxyribozyme oscillators RC. The results of the NRMSE of the short-term memory task and the long-term memory task are shown in Fig. 8 and Fig. 9, respectively. We ran 10 simulations for each setting with different input hold times τ and base influx rates. The input hold time τ was swept from 0.2 to 0.5 with a 0.1 increment. This allows us to look at the system with different number of perturbations, specifically from 2 perturbations to 5 perturbations. The base influx rate was swept from 0.0001 to 0.0005 with a 0.0001 increment. The range of the base influx rate sweeping was chosen based on the TPM and genetic search.

Model Comparison Fig. 10 shows that the random DNA strand circuit RC achieves 76.5% and 53.6% better performance than the deoxyribozyme oscillator RC in Goudarzi *et al.* [2] for the short-term and long-term memory task, respectively. Furthermore, this RC model achieves 68.2% and 60.8% improvement in performance comparing to the deoxyribozyme oscillator RC in Yahiro *et al.* [3]. In particular, the deoxyribozyme oscillator RC in Goudarzi *et al.* achieves 0.23 ± 0.05 and 0.11 ± 0.02, the deoxyribozyme oscillator RC in Yahiro *et al.* achieves 0.17 ± 0.034 and 0.13 ± 0.036, and the random DNA strand circuit RC achieves 0.054 ± 0.024 and 0.051 ± 0.026, in NRMSE for short-term and long-term memory task, respectively.

However, the most important trade-off of using the random DNA strand circuit RC is the cost. The random DNA strand circuit RC model requires 17 species, while the deoxyribozyme oscillator RC models only require 3 species. Hence, the system is 82% larger in size.

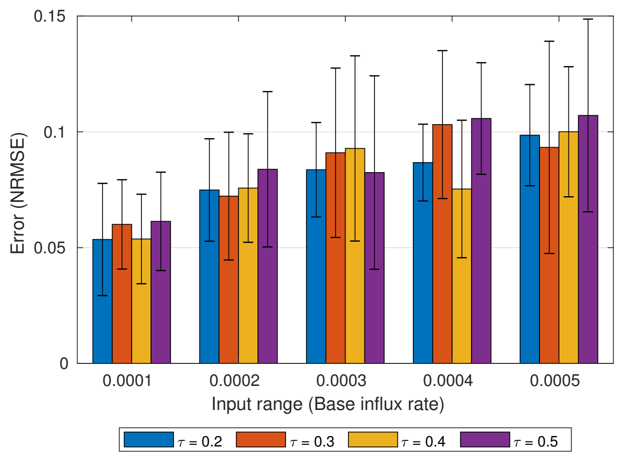

Fig. 8. NRMSE of the output of the short-term memory task, learned by the random DNA strand circuit RC. The reservoir achieves the best performance (lowest NRMSE = 0.054 ± 0.024) when the base influx rate is 0.001 and input hold time $\tau = 0.2$. The whiskers represent the standard deviation over 10 simulations. This can be reduced by averaging over a larger number of simulations.

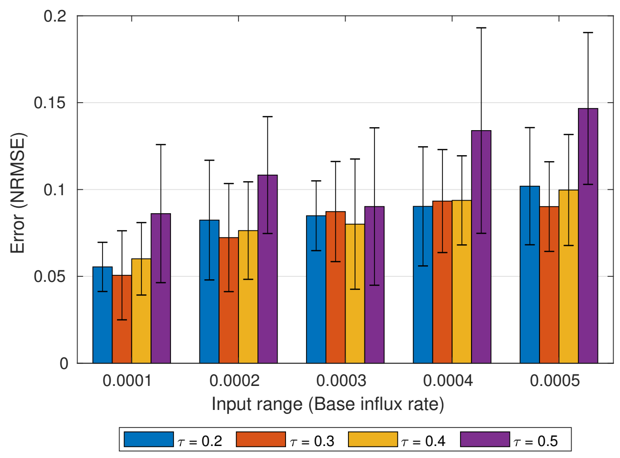

Fig. 9. NRMSE of the output of the long-term memory task, learned by the random DNA strand circuit RC. The reservoir achieve the best performance (NRMSE = 0.051 ± 0.026) when the base influx rate is 0.001 and input hold time $\tau = 0.3$. The whiskers represent the standard deviation over 10 simulations. This can be reduced by averaging over a larger number of simulations.

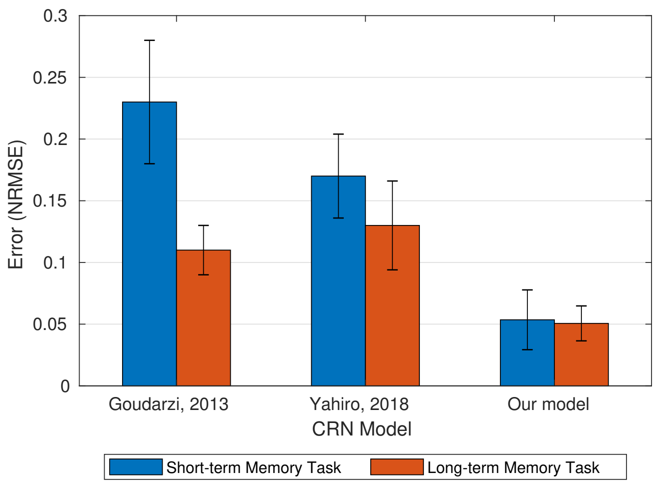

Fig. 10. Performance comparison of the best performance setups between the deoxyribozyme oscillator RCs and the random DNA strand circuit RC. The random DNA strand circuit RC has its performance improved 76.5% and 53.6% compared to the deoxyribozyme oscillator RC in Goudarzi *et al.* for the short-term and long-term memory task, respectively. Also, the performance of our RC model improves 68.2% and 60.8% in NRMSE compared to the deoxyribozyme oscillator RC in Yahiro *et al.*

4 Conclusion and Future Work

In this paper, we have introduced and simulated a stochastic model of a random DNA strand circuit. The reaction dynamics of the chemical species serving as a reservoir kernel allowed the system to learn two temporal computing tasks. Compared with the state of the art [2,3], our RC embedded in a random DNA strand circuit achieved better performance on the short- and long-term memory tasks. Specifically, the random DNA strand circuit RC outperforms the deoxyribozyme oscillator RC in Goudarzi *et al.* by 76.5% and 53.6%, and in Yahiro *et al.* by 68.2% and 60.8%, for short-term memory task and long-term memory task, respectively. Further, our random DNA strand circuit RC can be trained to learn the Hamming distance between two input bitstreams, with the average learning NRMSE of 0.18 ± 0.38, over four different input hold times.

Our novel random DNA strand architecture has the potential to simplify the way we build adaptive biomolecular circuits. Such circuits have applications in the area of biomedical diagnosis, pathogen detection, and industrial monitoring.

In future work, we will prove the robustness and the usefulness of the stochastic behavior by testing the random DNA strand circuit RC to detect pathogen and gene mutations. We also plan to verify the simulation results with actual DNA strands.

5 Acknowledgements

This material is based upon work supported by the National Science Foundation under grants no. 1518833 and 1518861.

References

1. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry **81**(25), 2340–2361 (1977)
2. Goudarzi, A., Lakin, M., Stefanovic, D.: DNA Reservoir Computing: A Novel Molecular Computing Approach. In: DNA Computing and Molecular Programming, pp. 76–89. Springer International Publishing, Cham (2013)
3. Yahiro, W., Aubert-Kato, N., Hagiya, M.: A reservoir computing approach for molecular computing. The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE) (30), 31–38 (2018)
4. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences **107**(12), 5393–5398 (2010)
5. Dittrich, P., Ziegler, J., Banzhaf, W.: Artificial Chemistries - A Review. Artificial Life **7**(3), 225–275 (2001)
6. Arceo, C., Jose, E., Marin-Sanguino, A., Mendoza, E.: Chemical reaction network approaches to Biochemical Systems Theory. Mathematical Biosciences **269**, 135–152 (2015)
7. Schrauwen, B., Verstraeten, D. Van Campenhout, J.: An overview of reservoir computing: theory, applications and implementations. In: 15th European Symposium on Artificial Neural Networks, pp. 471–482. (2007)
8. Farfel, J., Stefanovic, D.: Towards Practical Biomolecular Computers Using Microfluidic Deoxyribozyme Logic Gate Networks. In: DNA Computing, pp. 38–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
9. Morgan, C., Stefanovic, D., Moore, C., Stojanovic, M.": Building the Components for a Biomolecular Computer. In: DNA Computing, pp. 247–257. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
10. Penrose, R.: A generalized inverse for matrices. Mathematical Proceedings of the Cambridge Philosophical Society **51**(3), 406–413 (1955)
11. Zadeh, J., Steenberg, C., Bois, J., Wolfe, B., Pierce, M., Khan, A., Dirks, R., Pierce, N.: NUPACK: Analysis and design of nucleic acid systems. Journal of Computational Chemistry **32**(1), 170–173 (2011)
12. Szathmáry E.: The origin of replicators and reproducers. Philosophical transactions of the Royal Society of London. Series B, Biological sciences **361**(1474), 1761–1776 (2006)
13. Segré, D., Lancet, D., Kedem, O., Pilpel, Y.: Graded Autocatalysis Replication Domain (GARD): Kinetic Analysis of Self-Replication in Mutually Catalytic Sets. Origins of life and evolution of the biosphere **28**(4), 501–514 (1998)
14. Stadler, P., Fontana, W., Miller, J.: Random Catalytic Reaction Networks. Phys. D **63**(3-4), 378–392 (1993)
15. Blount, D., Banda, P., Teuscher, C., and Stefanovic D.: Feedforward Chemical Neural Network: An In Silico Chemical System That Learns XOR. Artificial Life 23:3, 295–317 (2017)
16. Banda, P., Teuscher, C., and Lakin M. R.: Online Learning in a Chemical Perceptron. Artificial Life 19:2, 195–219 (2013)
17. Banda, P., Teuscher, C., and Stefanovic D.: Training an asymmetric signal perceptron through reinforcement in an artificial chemistry, Journal of The Royal Society Interface 11:93,20131100 (2014)

18. Lakin, M. R., Minnich, A., Lane, T., and Stefanovic, D.: Design of a biochemical circuit motif for learning linear functions. Journal of The Royal Society Interface 11:101, 20140902 (2014)
19. Banda, P., and Teuscher, C.: An Analog Chemical Circuit with Parallel-Accessible Delay Line for Learning Temporal Task. In: ALIFE 14: The Fourteenth Conference on the Synthesis and Simulation of Living Systems 14, 482–489 (2014)
20. Lakin, M. R. and Stefanovic, D.: Supervised learning in an adaptive DNA strand displacement circuit. In: DNA Computing and Molecular Programming, Lecture Notes in Computer Science 9211, 154–167 (2015)
21. Lakin, M. R. and Stefanovic, D.: Supervised learning in adaptive DNA strand displacement networks. ACS Synthetic Biology 5:8, 885–897 (2016)
22. Ouyang, Q.: DNA Solution of the Maximal Clique Problem. Science **278**(5337), 446–449 (1997)
23. Faulhammer, D., Cukras, A., Lipton, R., Landweber, L.: Molecular Computation: RNA Solutions to Chess Problems. Proceedings of the National Academy of Sciences of the United States of America **97**(4), 1385–1389 (2000)
24. Stojanovic, M., Stefanovic, D.: A Deoxyribozyme-based Molecular Automaton. Nature Biotechnology **21**(9), 1069–1074 (2003)
25. Woods, D., Doty, D., Myhrvold, C., Hui, J., Zhou, F., Yin, P., Winfree, E.: Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. Nature **567**, 366–372 (2019)
26. Qian, L., Winfree, E.: Scaling up digital circuit computation with DNA strand displacement cascades. Science **322**, 1196–1201 (2011)
27. Hordijk, W., Steel, M.: Autocatalytic Networks at the Basis of Lifes Origin and Organization. Life **8**(4), 62 (2018)
28. Ashkenasy, G., Jagasia, R., Yadav, M., Ghadiri, M.: Design of a directed molecular network. Proceedings of the National Academy of Sciences of the United States of America **101**(30), 10872–10877 (2004)
29. Lukoševičius, M., Jaeger, H.: Reservoir Computing Approaches to Recurrent Neural Network Training. Computer Science Review **3**(3) 127–149 (2009)
30. Schrauwen, B., Büsing, L., Legenstein, R.: On computational power and the order-chaos phase transition in reservoir computing. In: Advances in Neural Information Processing Systems 21 (NIPS 2008), pp. 1425–1432. (2009)
31. Büsing, L., Schrauwen, B., Legenstein, R.: Connectivity, Dynamics, and Memory in Reservoir Computing with Binary and Analog Neurons. Neural Computation **22**(5), 1272–1311 (2010)
32. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation **14**(11), 2531–2560 (2002)
33. Fernando, C., Sojakka, S.: Pattern Recognition in a Bucket. In: Advances in Artificial Life, pp. 588–597. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
34. Jones, B., Stekel, D., Rowe, J., Fernando, C.: Is there a Liquid State Machine in the Bacterium Escherichia Coli? In: 2007 IEEE Symposium on Artificial Life, pp. 187–191. (2007)