A reservoir computing approach for molecular computing

Wataru Yahiro[1], Nathanael Aubert-Kato[2,3], Masami Hagiya[1]

[1]Department of Computer Science, the University of Tokyo
[2]Department of Information Science, Ochanomizu University
[3]Institute of Industrial Sciences, the University of Tokyo
naubertkato@is.ocha.ac.jp, hagiya@is.s.u-tokyo.ac.jp

Abstract

In this paper, we apply the Polymerase-Exonuclease-Nickase Dynamic Network Assembly (PEN DNA) toolbox, a modular framework for molecular computing, to reservoir computing. While reservoir computing is traditionally implemented with recurrent neural networks, any system with similar recurrent properties, here chemical reaction networks (CRNs), can be used as a reservoir.

We compared our approach to a previous CRN implementation of reservoir computing by Goudarzi et al. Our implementation yielded similar performance with respect to their benchmark tasks.

We then took advantage of the modularity of the PEN DNA toolbox to investigate the impact of the CRN size on performance, both by hand and with an automated optimization process. In both cases, we were able to find systems with excellent performance while also being realistic with respect to *in vitro* implementation.

Finally, we investigated the impact of constraining the weights of the output layer to be positive. This constraint guarantees that the system will remain relatively small, and thus makes it easier to implement *in vitro*. While this constraint led to an expected degradation in performance, we were still able to find good implementations of the reservoir.

Introduction

Ever since the seminal work of Adleman (Adleman, 1994), molecular computing has proven to be a powerful and flexible approach to programming systems *in vitro*. DNA molecules have been used extensively for designing CRN because they undergo reasonably predictable reactions, including hybridization and denaturation (see, for instance, the review by Padirac et al. (2013)).

Though the focus of the original studies was limited to problem-specific implementations (Hagiya, 2004), many general frameworks have been proposed since then (Seelig et al., 2006; Qian et al., 2010; Montagne et al., 2011; Weitz and Simmel, 2012).

In contrast to classical, "digital" computers, we must use analog programming when working with molecular computers. The non-linear nature of this process raises the question of how to design and implement complex systems by chemical reactions, which is one of the central research topics in this field. This is even more difficult when designing temporal systems that are expected to react continuously to time-series inputs, such controllers for molecular robots, rather than systems that perform a single computation.

Current efforts have focused on either computer-assisted design approaches (Lakin et al., 2011; Aubert et al., 2014), or black-box optimization strategies (Kawamata and Hagiya, 2016; Dinh et al., 2015). Computer-assisted design approaches provide easy interfaces for describing molecular systems that can then be simulated or even verified. This process allows researchers to quickly iterate between different versions of a system, thus enabling them to design by trial and error. On the other hand, black-box optimization strategies aim to solve the inverse problem: given a target behavior, they search for systems that implements it. In this case, the user only defines the way to evaluate the performance of the system, and then the algorithm generates candidate solutions until it finds a good match. The term "black-box optimization" refers to the fact that knowledge of the inner workings of candidate solutions is unnecessary from the algorithm's point of view. Thus, the solutions are considered black boxes.

In this paper, we investigate a different strategy called *reservoir computing*, which is an implementation of recurrent neural networks where most of the network is held fixed (the reservoir) and the learning approach is constrained to one layer (the read-out) (Lukoševičius and Jaeger, 2009; Maass et al., 2002; Jaeger, 2002). Our approach is motivated by the recent findings that chemical systems can be used as reservoirs (Goudarzi et al., 2013; Jones et al., 2007).

In this paper, we first evaluate the performance of a PEN DNA toolbox oscillator with the same structure as that of Goudarzi et al. We also add a new constraint, namely that the output layer is positively weighted. DNA computing-based implementations of addition and multiplication are relatively easy to realize (Kobayashi et al., 2014), but subtraction is very difficult. Therefore, output layers consisting only of positive weights is desirable because they are easier to im-

plement *in vitro*.

While Goudarzi et al. studied an oscillator limited to three gates, we took advantage of the modularity of the PEN DNA toolbox by implementing oscillators with different sizes and structures and investigating their performance as reservoirs.

Finally, we applied ERNe (Evolving Reaction Network, Dinh et al. (2015)), a framework for evolving PEN DNA toolbox systems, to search for reservoirs with high performance. We used that performance as fitness in the evolutionary optimization.

Related Work

Reservoir Computing

Reservoir computing is one of several approaches to designing and training recurrent neural networks (Lukoševičius and Jaeger, 2009). Recurrent neural networks contain loop structures, which help with the handling of time-series data. The concept of reservoir computing is based on the idea that dynamical systems that exhibit sufficiently complex behavior should have the capacity for memory and computation.

The reservoir of a reservoir computing system is implemented in the intermediate layer, which is a complex recurrent neural network. The output layer, which is known as the readout, is the only component of the whole network to be updated during the learning phase. The speed of the process is increased significantly by fixing the weights of the input and intermediate layers, thus limiting the number of weights to be trained. Note that otherwise, the learning process proceeds according to standard neural network approaches.

Though reservoir are usually implemented as (recurrent) neural networks, their fixed nature means that they can be considered black boxes. Hence, this component can be replaced by any system that has the properties of a reservoir, including physical systems (see, for instance, the "exotic" reservoirs in Lukoševičius and Jaeger (2009)). As such, reservoir computing is also relevant to the fields of natural and unconventional computing.

Deoxyribozyme Oscillator as a Reservoir Goudarzi, et al. analyzed a reservoir implemented by a DNA reaction system called a deoxyribozyme oscillator (Goudarzi et al., 2013). A deoxyribozyme oscillator consists of three NOT gates, each of which is implemented by a deoxyribozyme gate (Stojanovic and Stefanovic, 2003). The connections between the gates are arranged such that they deactivate each other in a rock-paper-scissors fashion. Goudarzi et al. used the open flow reactor proposed by Morgan et al. (Morgan et al., 2004) to provide a continuous supply of fuel, which is necessary to sustain the operation of the system while preventing the accumulation of chemical waste.

They used one of the molecular influxes as the time-series input of the whole system, varying it over time. The resulting unbalance affected the behavior of the oscillator. The concentrations of chemical wastes produced by each gate (and the concentrations of unused gates) were taken as the outputs of the reservoir. They then multiplied these outputs by the output weights matrix W^{out} to obtain the output $Y(t)$ of the entire system.

They analyzed their model by assessing its performance with respect to two tasks, which we also use as benchmarks, testing (A) the short term memory storage and (B) the long term memory storage. The target output $\hat{Y}(t)$ of both tasks was defined as a function of time, which was calculated from the time-series input to the entire system.

They then minimized the difference between $Y(t)$ and $\hat{Y}(t)$ by linear regression with respect to W^{out}. Finally, they evaluated their results in terms of the normalized root-mean-square error (NRMSE).

While their results were promising, it is difficult to extend the design of the deoxyribozyme oscillator, which limits their system in terms of its potential application to more complex functions. Additionally, Goudarzi et al. allowed their output weights matrix to contain negative components. We believe that negative output weightings should be avoided to keep the system as simple as possible because it is difficult to implement subtraction with molecular reactions. (Kobayashi et al., 2014).

Gene Regulation Network as a Reservoir Jones, et al. reported that it may be possible to use a cell's gene regulation network as a reservoir (Jones et al., 2007). They used a rough model of the E. coli gene regulation network, taking the concentrations of mRNA and proteins as the outputs from the reservoir and implementing the readout layer as multiple perceptrons. The concentrations of two inhibitor proteins were used as time-series inputs.

Although the reservoir performed well at simple tasks, such as performing a XOR, they reported that the dimensionality of the gene regulatory network reservoir was relatively low, and it exhibited limited recurrence. This suggests that more complex networks may be required. The PEN DNA toolbox makes it possible to implement such networks as reaction networks.

PEN DNA Toolbox

The PEN DNA toolbox (Montagne et al., 2011; Padirac et al., 2012) is a molecular programming framework that can be used to implement reaction networks with arbitrary sizes. Moreover, it is relatively easy and inexpensive to implement those systems *in vitro* (Baccouche et al., 2014).

The PEN DNA toolbox offers three base modules: activation, inhibition and autocatalysis. These modules can be combined to build complex networks. Moreover, chemical species produced are continuously degraded through enzymatic activity, which prevents accumulation. Those modules are implemented by DNA species called templates, which are chemically protected against degradation. This ensures that their concentrations remain stable over time.

It is easy to represent each of the modules of the PEN DNA toolbox graphically, so the reaction networks obtained have straightforward graphical representations (Fig. 1).

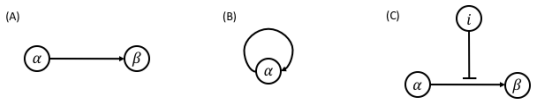

Figure 1: The graphical representation of (A) activation, (B) autocatalysis and (C) inhibition. The nodes represent signal species, dynamically produced and degraded by the system. The arrows represent chemical species called templates which produce a output when the corresponding input is present. The bar-headed arrows represent inhibition, which temporarily disables activation until the inhibiting sequence is degraded.

Methods

Formulation of Reservoir Computing

The reservoir computing model consists of three layers: input, intermediate (reservoir) and output (read-out). Let I and O denote the number of inputs and outputs respectively, and assume that the reservoir is a directed graph with N nodes. Additionally, edges connect the input layer to the reservoir, which in turn is connected to the output layer. The weights of these edges are defined in two matrices. Let $W^{\text{in}} = [w^{\text{in}}_{ij}]$ be the $N \times I$ matrix of input weights, where w^{in}_{ij} is the weight of the connection between input node i and reservoir node j. The weights of the connections inside the reservoir are given by the $N \times N$ matrix $W^{\text{res}} = [w^{\text{res}}_{jk}]$, where w^{res}_{jk} is the weight from node j to node k. Finally, let $W^{\text{out}} = [w^{\text{out}}_{kl}]$ be the $O \times N$ output weight matrix, where w^{out}_{kl} is the weight of the connection between reservoir node k and output node l. The I-dimensional vector $U(t) = [u_i(t)]$ represents the time-series input; the state of the reservoir is specified by the N-dimensional vector $X(t) = [x_j(t)]$; and the output of the system is the O-dimensional vector $Y(t) = [y_k(t)]$.

At each time-step, the internal state of the reservoir is updated by:

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

where f is a nonlinear function. The output of the reservoir is calculated by taking the linear combination of the state vector:

$$Y(t) = W^{\text{out}} \cdot X(t) + w_b \tag{2}$$

where w_b is an inductive bias.

The matrices W^{in} and W^{res} are fixed because the reservoir is treated as a black box. For a given output $\hat{Y}(t)$, we train the output weight matrix W^{out} to minimize the error $||Y(t) - \hat{Y}(t)||^2$. This is a simple linear regression problem that can be solved by the Moore-Penrose pseudo-inverse method (Penrose, 1955):

$$W^{\text{out}'} = (X'^T \cdot X')^{-1} \cdot X'^T \cdot \hat{Y}' \tag{3}$$

where $W^{\text{out}'}$ is the trained output matrix extended with inductive biases, and X' is the observation matrix of reservoir, each row of which is obtained by adding a constant term, 1, to the state vector of the reservoir at each time point. $\hat{Y}'$ is the target output matrix, which rows represent the target output at each time point.

Non-negative constraint

We take the final goal of molecular robotics into account by imposing the constraint that molecular weights must be positive. This makes the computation of linear combinations of molecular concentrations easily feasible *in vitro* (Kobayashi et al., 2014).

Specifically, this constraint forces all the weights in the trained read-out layer to be positive. Learning becomes a linear regression problem, which we solve by the active set method (Bro and De Jong, 1997). We implemented this approach using the scipy function `scipy.optimize.nnls`.

Tasks

We evaluated the performance of our reservoirs using the same tasks, Task A and Task B, as used in (Goudarzi et al., 2013).

Our goal was for the concentrations of the species at each time to approximate the values of the target functions $\hat{Y}(t)$. The value of the output $Y(t)$ is defined by Equation 2. The target function, $\hat{Y}(t)$, of Task A (short term memory) is defined as follows:

$$\hat{Y}(t) = S^m_1(t-1) + 2S^m_1(t-2) \tag{4}$$

which is calculated based on the inputs from 1 and 2 minutes before. We defined the target function for Task B, $\hat{Y}(t)$ (long term memory) as:

$$\hat{Y}(t) = S^m_1(t-\tau) + \frac{1}{2}S^m_1(t-\frac{3}{2}\tau) \tag{5}$$

Thus, the system is required to output the inputs from $\tau[\text{min}]$ and $\frac{3}{2}\tau[\text{min}]$ ago to successfully complete the task.

Input

Like Goudarzi et al., we encoded the input as an influx of one of the molecular species (indicated as S_1 in the chemical reaction networks). We defined the input as $S^m_1(t) = SR$, where S is the maximum possible value for the input and R is a random real between 0 to 1. The input function varies randomly at discrete intervals of length τ. We gave the system time to reach its stable oscillatory regime by delaying the input until $t = 1,000[\text{min}]$.

Performance evaluation of the system

We evaluated the performance of the system by calculating the error between the output $Y(t)$ generated by the trained weights and the target function $\hat{Y}(t)$. We evaluated the error by calculating the value of the NRMSE, defined as follows:

$$\text{NRMSE} = \frac{1}{Y_{\max} - Y_{\min}} \sqrt{\frac{\sum_{t=t_1}^{t_n} (\hat{Y}(t) - Y(t))^2}{n}} \quad (6)$$

where $Y_{\max}$ and $Y_{\min}$ are, respectively, the maximum and minimum output values $Y(t)$. Note that the output $Y(t)$ matches $\hat{Y}(t)$ perfectly when NRMSE $= 0$.

Molecular reservoirs

Basic designs First, we considered two simple fixed patterns with variable sizes: oscillators and lines.

An n-oscillator is composed of n signal-inhibitor pairs, so that its size is $2n$ with respect to the number of species (Figure 2, left). Note that n needs to be odd for the system to actually oscillate.

An n-line consists of n signals, chained by templates (Figure 2, left). While this does not define a recurrent network per se, the transcription delay between nodes provides the system with a transient memory.

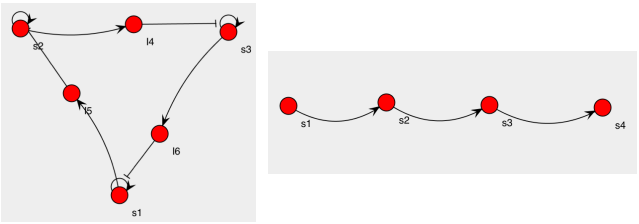

Figure 2: Structure of a 3-oscillator (left) and a 4-line (right).

Simulation We employed and extended an existing simulator for PEN DNA toolbox systems called DACCAD (Aubert et al., 2014) to carry out the numerical work in this study. We used DACCAD as a library for generating and solving the differential equations needed to simulate a given PEN DNA toolbox reaction system.

Reservoir evolution Since the behavior of reaction networks is highly non-linear (Genot et al., 2016) and thus unintuitive, it is difficult to design reaction networks that are suitable for specific purposes.

We automated the discovery of good reservoirs using another framework for evolving reaction networks on PEN DNA toolbox systems (Dinh et al., 2015; Aubert-Kato et al., 2017), called Evolving Reaction Network (ERNe, also known as bioNEAT), which is an implementation of an evolutionary algorithm for optimizing both the topology and the parameters of PEN toolbox networks. ERNe is based on NEAT (Stanley and Miikkulainen, 2002), an algorithm designed to evolve neural networks, with extensions to deal with chemical properties such as sequence stability and PEN DNA toolbox inhibition.

Candidate solutions (graphs) generated by the ERNe framework are evaluated by a fitness function, and those with higher fitness scores are kept (with mutations) in the next evaluation round. Here, the fitness is the inverse of the RMSE:

$$\text{fitness} = \frac{1}{\text{RMSE}} = \sqrt{\frac{n}{\sum_{t=t_1}^{t_n} (\hat{Y}(t) - Y(t))^2}} \quad (7)$$

We fixed the input parameters to $S = 0.25$ [nM/ min] and $\tau = 50$ [min]. We ran 10 runs of 50 generations, with 50 candidate solutions per generation, under both normal and non-negative conditions.

Results

Performance Analysis of 3-Oscillator

For our first step, we compared the deoxyribozyme oscillator experimental setting (Goudarzi et al., 2013) to a similar network implemented with the PEN DNA toolbox (Figure 2, left). Oscillations follow the following process: when a large amount of a given autocatalyst is present, it produces enough inhibitor to kill the next autocatalyst in the cycle, stopping the inhibition of the third autocatalyst. That autocatalyst then increases in concentration, eventually killing the first autocatalyst and completing the cycle.

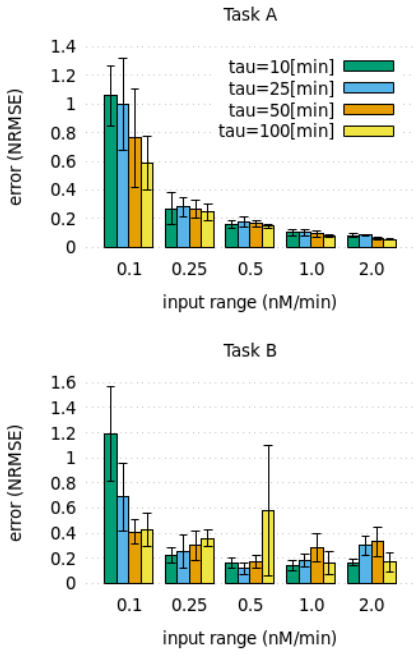

Figure 3: The average and standard deviation of the NRMSE of a 3-oscillator for Task A (top) and Task B (bottom).

We ran 10 simulations for each pair of delays $\tau = 10$, 25, 50 and 100 [min], and maximum influx $S = 0.1$, 0.25, 0.5, 1.0 and 2.0 [nM/ min]. We then calculated the average and standard deviation of the NRMSE after training. The results are shown in Figure 3.

The performance at Task A (Figure 3, top) clearly improved as the input range S increased. This was probably because the entire system was more affected by the current input when the input was larger. On the other hand, the performance should have improved as the interval τ increased because the target function $\hat{Y}$ fluctuated less. Although we did observe this tendency, its influence did not seem to be as large as that of S.

No clear trend was observed for Task B (Figure 3, bottom), even though, in theory, the parameters should have had a similar impact as observed in Task A. The system needed to store more information as the length of the interval τ increased, thus making the task harder. The performance of the system with the shortest interval $\tau = 10$ [min]. improved with increasing S. In other cases, when $\tau = 25$ and 50 [min], the best performance was obtained when $S = 1.0$ [nM/ min]. In the case of $\tau = 100$ [min], the NRMSE was unexpectedly large when $S = 0.5$ or 2.0 [nM/ min] while it was minimized when $S = 1.0$ [nM/ min]. This means that S has an optimal input range for a target amount of memory.

The best performance at Task B was obtained when $S = 0.5$ [nM/ min] and $\tau = 25$ [min]. Figure 4 shows the concentrations of species over time.

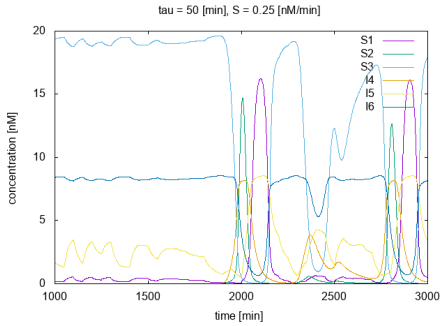

Figure 4: The concentrations of species in the system with error minimizing parameters. Note that those parameters keep the system near the oscillatory regime, a type of behavior called 'excitatory'.

The values of the NRMSE for Task A and Task B were 0.17 ± 0.034 and 0.13 ± 0.036, respectively. Figure 5 shows the target and output for both tasks. The NRMSEs of the deoxyribozyme oscillator Goudarzi et al. (2013) were 0.23 and 0.11, respectively, for Task A and Task B. As such, the performance of the PEN toolbox 3-oscillator was similar to that of the deoxyribozyme oscillator.

The behavior of the entire system depended on the range

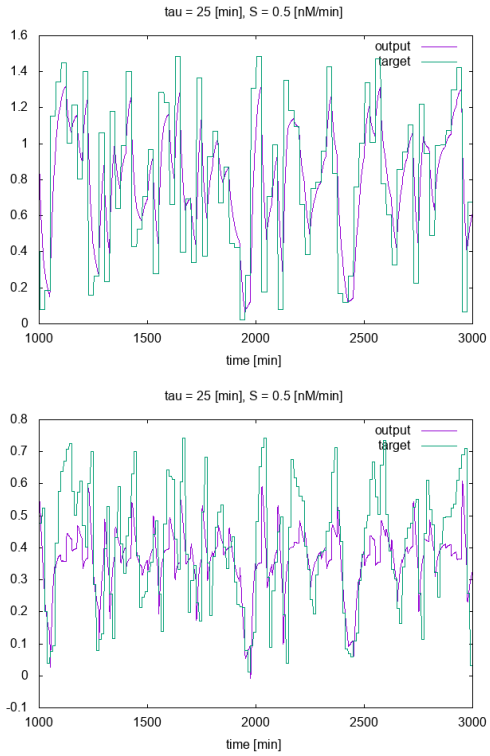

Figure 5: Target (green) and output (purple) values of Task A (top) and Task B (bottom)

of the input S. When $S = 0.1$ [nM/ min], the input was too weak to affect the behavior of the system. On the other hand, when $S = 2.0$ [nM/ min], the concentrations of output species are very similar to that of the input, meaning they were directly controlled by the input. Thus, the system could not read information when the input was too weak, and the state of the entire system was overwritten by the input when it was too strong. Thus, both cases yielded bad reservoirs.

This implies that the range of valid inputs may be limited in practice, even when we use a good network topology. In particular, we need to find a system with a valid input concentration range that is large enough to allow for noise.

Analysis of the Performance under the Non-negative Constraint

We used the same range of parameters and settings in this case as in the standard case, the results of which are shown in Figure 6.

The performance for both tasks with respect to S and τ varied in a similar manner to that of the standard case. However, the performance was degraded. This is a natural consequence of the fact that restricting the elements of the weights matrix to take only non-negative values limits the scope for optimization.

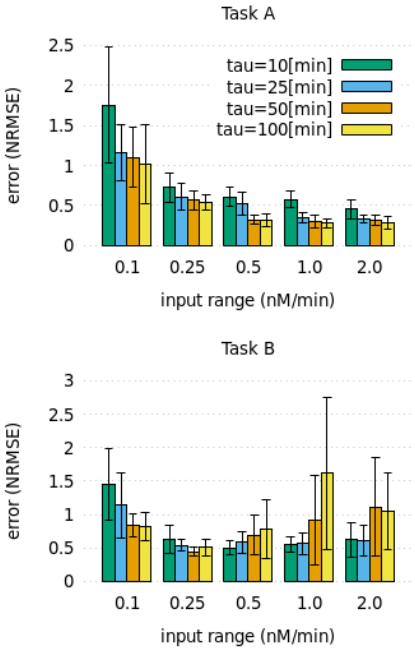

Figure 6: The average and standard deviation of the NRMSE of a 3-oscillator under the non-negative constraint for Task A (top) and Task B (bottom).

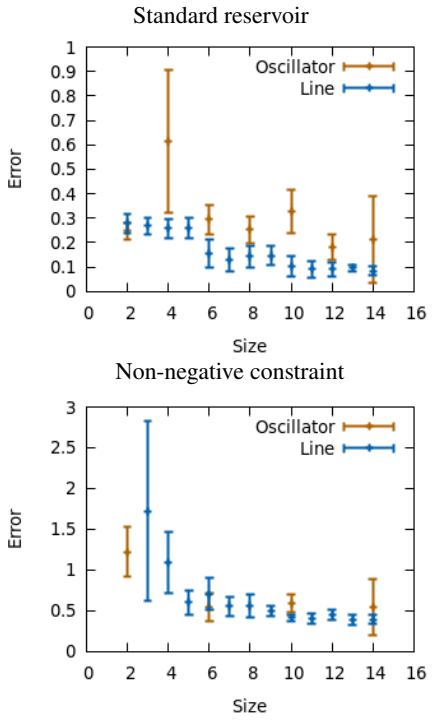

Figure 7: The relationships between graph size, graph type, and performance.

Impact of graph size

A simple way to increase the complexity of a reservoir is to scale up the size of its network. Doing so also increases the size of the output layer, giving more leeway for training.

We investigated the impact of size on the performance of PEN toolbox reservoirs for both basic types of graph: oscillators and lines. We tested Task B (long term memory) for graph sizes ranging from $n = 2$ to 14. The parameters were fixed to $S = 0.25$ [nM/ min] and $\tau = 50$ [min] We ran the simulation and calculated the mean and standard deviation of the NRMSE over 10 trials for each graph.

The results obtained by training the weight vector via linear regression, under both normal and non-negative constraints, are shown in Figure 7. Note that the scales of the vertical axes differ significantly between the two cases.

Figure 7, top, shows the results for oscillators and lines when trained under normal learning conditions. Increasing the number of nodes in the oscillator did not lead to any significant improvement, and the NRMSE fell between 0.2 and 0.4 in most cases.

The performance of the line graphs increased significantly between $n = 5$ and 6. This may either be because the system became large enough to store the input information or long enough for transduction to induce a natural delay. Note that those two phenomena can be considered equivalent from the perspective of reservoir computing. More sustain is available to longer graphs, thus making it easier to maintain the memory. The NRMSE was 0.090 ± 0.0067 when $n = 14$.

Figure 7, bottom, shows the clear trend in the results under the non-negative constraint. The NRMSE was relatively small, between 0.4 to 0.7, when $n = 3, 5, 7$ (number of nodes are $6, 10$ and 14), in the case of the oscillator. The NRMSE was extremely large (8.8 ± 9.8, 37.0 ± 53.8 and 19.6 ± 20.2) when $n = 2, 4$ and 6 (corresponding to 4, 8 and 12 nodes in total). These data points were thus excluded from the Figure. The difference is clearly related to the fact that systems containing cycles with even n are multi-stable. These would, in general, be considered as systems with memory (Padirac et al., 2012); however, in our case, the input biased the system towards being in a state producing S_1, so the output did not change noticeably.

The performance of the line graphs increased exponentially until $n = 5$, then improved linearly as the graphs increased further in size, with an NRMSE of 0.37 ± 0.030 when $n = 14$. This is comparable to the NRMSE of the 3-oscillator obtained by normal linear regression training (NRMSE $= 0.34 \pm 0.070$).

These results show that the reservoir should be large enough to memorize information about the input. However, a large size is not enough because the specific dynamics of the graph, such as multi-stability, can greatly affect the performance.

Evolving reservoirs

We applied evolutionary optimization to improve the performance beyond the plateau reached by the basic graphs. As before, we performed training under both normal and non-negative linear regression conditions. We focused on Task B (long term memory) because it was the harder of the two.

The performance of every individual with respect to its size is plotted in Figure 8. We can see that the performance increased quickly for small graphs, then plateaued before a final increase around 15 nodes for standard regression and 10 nodes for non-negative regression. We can also see that the performance actually tended to decrease beyond that point, possibly owing to the increase in the number of parameters to fit for the larger graphs. Simpler networks are both easier to implement *in vitro* and less subject to the reality gap. Hence, a multi-objective optimization between size and performance may yield even better results in the future.

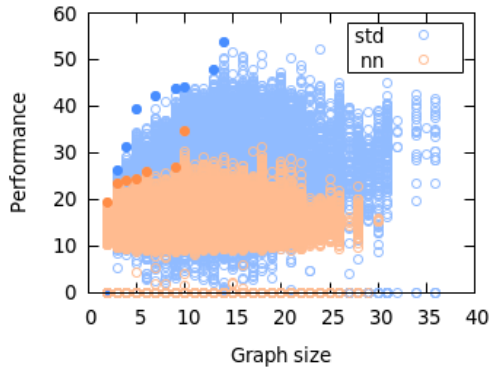

Figure 8: Distribution of individuals in the standard (std) and non-negative (nn) cases. The Pareto front is indicated with filled circles.

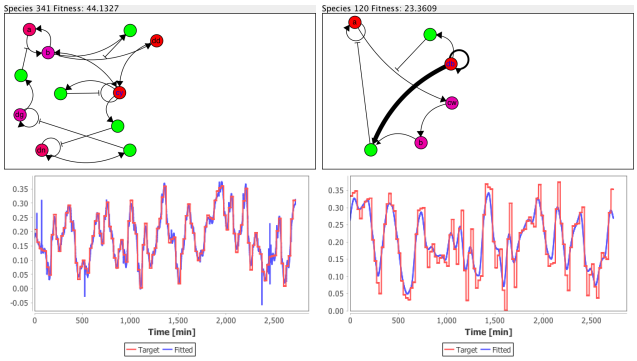

Figure 9: Best stable individual when using normal (left) and non-negative linear regression (right).

Figure 9 shows the best individuals among those in the last generation of each runs evolved under normal (left) and non-negative (right) constraints. While better performing systems were found, they were not maintained in the population. This indicates their lack of robustness to parameter modification, and means their *in vitro* would be difficult. Also note that these graphs are at the edge of the plateau in the Pareto front shown in Figure 8. The high fitness of systems beyond this front may be due to over-fitting. Their respective performances were 44.1 (NRMSE = 0.06) and 23.4 (NRMSE = 0.14), both of which are better than those of the rationally designed systems presented so far.

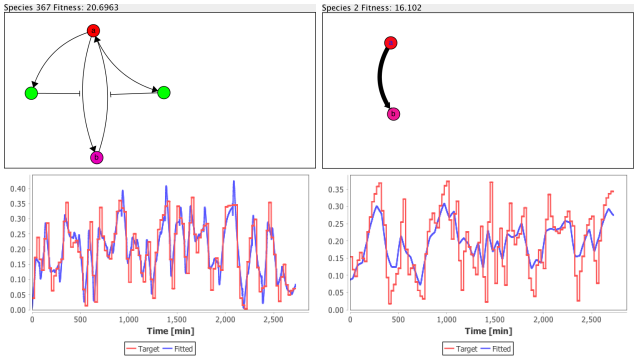

Figure 10: Good individuals with simple structures, evolved under normal (left) and non-negative constraints.

Systems at the beginning of the Pareto front are also interesting, as they performed reasonably while being simple enough to avoid the reality gap (Figure 10). The 2-line even achieved good performance with the right parameters.

Conclusion

In this paper, we proposed a reservoir computing implementation based on the PEN DNA toolbox. We found that approach effective, since non-linearities in toolbox systems cause them to behave like black boxes. First, we compared the performance obtained by systems constructed according to our approach to that of the molecular reservoir proposed by Goudarzi et al., with similar performance.

Being modular, the PEN DNA toolbox allowed us to investigate the performance of two classes of network with respect to their size: cycle oscillators and line graphs. As expected, larger networks achieved better performance, as the system could store more memory.

However, reaction networks may not be affected by their input if their internal dynamics are too strong. In particular, the system should not produce the input species to the point of saturation. Conversely, even some small networks with complex behaviors may perform much better as reservoirs.

We then searched for such reservoirs by applying an evolutionary method called ERNe. Multiple candidate solutions were found, including some composed of small networks, which were less likely to exhibit the reality gap.

We constrained the weights on the output layer to non-negative values, with the intention of improving the feasibility of implementing these systems *in vitro*. This constraint

means that the whole system has a straightforward chemical implementation. We found that imposing this constraint degraded the performance of the system with respect to the general case, but it remained acceptable for the purpose of molecular robotics.

Finally, the approach presented in this paper is relevant to the implementation of complex tasks, such as chemotaxis. While systems that perform these tasks can be obtained directly by designing reaction networks manually (Hagiya et al., 2016), the sheer size of the resulting networks makes the corresponding systems unrealistic and unresponsive. Smaller system should be able to compute changes in input concentration more quickly.

References

Adleman, L. M. (1994). Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024.

Aubert, N., Mosca, C., Fujii, T., Hagiya, M., and Rondelez, Y. (2014). Computer-assisted design for scaling up systems based on dna reaction networks. *Journal of The Royal Society Interface*, 11(93):20131167.

Aubert-Kato, N., Fosseprez, C., Gines, G., Kawamata, I., Dinh, H., Cazenille, L., Estevez-Tores, A., Hagiya, M., Rondelez, Y., and Bredeche, N. (2017). Evolutionary optimization of self-assembly in a swarm of bio-micro-robots. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 59–66. ACM.

Baccouche, A., Montagne, K., Padirac, A., Fujii, T., and Rondelez, Y. (2014). Dynamic dna-toolbox reaction circuits: a walkthrough. *Methods*, 67(2):234–249.

Bro, R. and De Jong, S. (1997). A fast non-negativity-constrained least squares algorithm. *Journal of chemometrics*, 11(5):393–401.

Dinh, H. Q., Aubert, N., Noman, N., Fujii, T., Rondelez, Y., and Iba, H. (2015). An effective method for evolving reaction networks in synthetic biochemical systems. *IEEE Transactions on Evolutionary computation*, 19(3):374–386.

Genot, A., Baccouche, A., Sieskind, R., Aubert-Kato, N., Bredeche, N., Bartolo, J., Taly, V., Fujii, T., and Rondelez, Y. (2016). High-resolution mapping of bifurcations in nonlinear biochemical circuits. *Nature chemistry*, 8(8):760.

Goudarzi, A., Lakin, M. R., and Stefanovic, D. (2013). DNA reservoir computing: a novel molecular computing approach. In *International Workshop on DNA-Based Computers*, pages 76–89. Springer.

Hagiya, M. (2004). Towards molecular programminga personal report on dna8 and molecular computing. In *Modelling in Molecular Biology*, pages 125–140. Springer.

Hagiya, M., Aubert-kato, N., Wang, S., and Kobayashi, S. (2016). Molecular computers for molecular robots as hybrid systems. *Theoretical Computer Science*, 632:4–20.

Jaeger, H. (2002). *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik.

Jones, B., Stekel, D., Rowe, J., and Fernando, C. (2007). Is there a liquid state machine in the bacterium escherichia coli? In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*, pages 187–191. Ieee.

Kawamata, I. and Hagiya, M. (2016). Design automation of nucleic acid reaction system simulated by chemical kinetics based on graph rewriting model. *Evolutionary Computation in Gene Regulatory Network Research*, pages 211–239.

Kobayashi, S., Yanagibashi, K., Komiya, K., Fujimoto, K., and Hagiya, M. (2014). Analog dna computing devices toward the control of molecular robots. In *Reliable Distributed Systems Workshops (SRDSW), 2014 IEEE 33rd International Symposium on*, pages 1–11. IEEE.

Lakin, M. R., Youssef, S., Polo, F., Emmott, S., and Phillips, A. (2011). Visual dsd: a design and analysis tool for dna strand displacement systems. *Bioinformatics*, 27(22):3211–3213.

Lukoševičius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Montagne, K., Plasson, R., Sakai, Y., Fujii, T., and Rondelez, Y. (2011). Programming an in vitro dna oscillator using a molecular networking strategy. *Molecular systems biology*, 7(1):466.

Morgan, C., Stefanovic, D., Moore, C., and Stojanovic, M. N. (2004). Building the components for a biomolecular computer. In *DNA*, pages 247–257. Springer.

Padirac, A., Fujii, T., and Rondelez, Y. (2012). Bottom-up construction of in vitro switchable memories. *Proceedings of the National Academy of Sciences*, 109(47):E3212–E3220.

Padirac, A., Fujii, T., and Rondelez, Y. (2013). Nucleic acids for the rational design of reaction circuits. *Current opinion in biotechnology*, 24(4):575–580.

Penrose, R. (1955). A generalized inverse for matrices. In *Mathematical proceedings of the Cambridge philosophical society*, volume 51, pages 406–413. Cambridge University Press.

Qian, L., Soloveichik, D., and Winfree, E. (2010). Efficient turing-universal computation with dna polymers. In *International Workshop on DNA-Based Computers*, pages 123–140. Springer.

Seelig, G., Soloveichik, D., Zhang, D. Y., and Winfree, E. (2006). Enzyme-free nucleic acid logic circuits. *science*, 314(5805):1585–1588.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Stojanovic, M. N. and Stefanovic, D. (2003). A deoxyribozyme-based molecular automaton. *Nature biotechnology*, 21(9):1069–1074.

Weitz, M. and Simmel, F. C. (2012). Synthetic in vitro transcription circuits. *Transcription*, 3(2):87–91.