A Comparison of Genetic Regulatory Network Dynamics and Encoding

Jean Disset
jean.disset@irit.fr

Dennis G Wilson
dennis.wilson@irit.fr

Sylvain Cussat-Blanc
sylvain.cussat-blanc@irit.fr

Stéphane Sanchez
stephane.sanchez@irit.fr

Hervé Luga
herve.luga@irit.fr

Yves Duthen
yves.duthen@irit.fr

University of Toulouse
IRIT - CNRS - UMR5505
Toulouse, France 31015

ABSTRACT

Genetic Regulatory Networks (GRNs) implementations have a high degree of variability in their details. Parameters, encoding methods, and dynamics formulas all differ in the literature, and some GRN implementations have a high degree of model complexity. In this paper, we present a comparative study of different implementations of a GRN and introduce new variants for comparison. We use a modified Genetic Algorithm (GA) to evaluate GRN performance on a number of common benchmark tasks, with a focus on real-time control problems. We propose an encoding scheme and set of dynamics equations that simplifies implementation and evaluate the evolutionary fitness of this proposed method. Lastly, we use the comparative modifications study to demonstrate overall enhancements for GRN models.

KEYWORDS

Genetic Algorithms, fitness evaluations, representations

ACM Reference format:
Jean Disset, Dennis G Wilson, Sylvain Cussat-Blanc, Stéphane Sanchez, Hervé Luga, and Yves Duthen. 2017. A Comparison of Genetic Regulatory Network Dynamics and Encoding. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071322

DOI: http://dx.doi.org/10.1145/3071178.3071322

1 INTRODUCTION

Artificial Genetic Regulatory Networks (GRNs) have varied in implementation since their conception, with earlier binary networks directly encoding connections giving way to proteins that interpolate their connection using exponential or other functions. Despite this diversity, there has been little attempt to compare the different aspects of GRN models.

Additionally, compared with other networked controllers, such as artificial neural networks (ANNs) and genetic programs (GP), GRNs have a number of complex implementation components. The protein encoding scheme and a concentration normalization step are focused on in this paper as possible points for simplification.

The complexity of implementation combined with the variety of GRN models in the literature can be a deterrent from the use of GRNs. This paper evaluates whether or not the GRN model can be simplified without losing performance quality. We use a set of signal processing and control tasks, both of which are common problem types for the GRN. Through our comparison of multiple GRN implementations, we propose individual improvements as well as a general best GRN model.

The next section § 2 details other works that use and study GRNs. An in-depth explanation of GRN models is given in § 3, and individual components of the GRN model that were studied in this work are explained in § 4. The details of the benchmark suite used are outlined in § 5, and the results of the experiments are presented in § 6. Lastly, the interpretation of these results and our plans for future work are detailed in § 7.

2 RELATED WORKS

The capability of GRNs has been displayed in a number of different domains. Joachimczak used GRNs in a robotic foraging problem [8], where a two-wheeled robot must collect randomly placed food particles and avoid poisonous ones. Joachimczak also used GRNs for signal processing [9], showing a GRN's capability to act as a signal amplifier or frequency filter.

Cussat-Blanc used GRNs to determine the optimal positions of turbines in a wind farm [15] and demonstrated competitive performance with SARSA on a suite of common reinforcement learning benchmarks [4], including mountain car, maze navigation, and acrobat. Nicolau originally showed the capabilities of a GRN on a single pole problem in [11].

GRNs have also been used in artificial embryogenesis contexts, similar to their original biological motivation. Cussat-Blanc used a cell-based model on the standard French flag problem [3] and developed mutli-cellular creatures from single cells with individual GRN controllers [6]. Lastly, in [16], GRNs were used to design ANNs, and [17] presented a mixed paradigm with spiking neural networks.

There has also been considerable study into the behavior and improvement of GRNs. GRN topology is examined in [7] to determine if evolved networks were scale-free and small-world, which are considered hallmarks of natural evolution. The timing dynamics of GRNs were studied in [10]. The benefits of variable-length GRNs are demonstrated in [14]. GRNEAT was presented in [5]; it is a GA specifically designed for GRNs and showed improvement on a number of tasks. GRNEAT is used in this paper and is further discussed in §§ 3.1.

3 GENETIC REGULATORY NETWORKS

The GRN described in this section was designed by Banzhaf in [1], with modifications made in [2]. The components under review in this study are further explored in § 4.

A GRN is composed of multiple artificial proteins, which interact via evolved properties. These properties, called tags, are

- The protein *identifier*, encoded as an integer between 0 and u_{size}. u_{size} can be changed in order to control the precision of the GRN.
- The *enhancer identifier*, encoded as an integer between 0 and u_{size}. The enhancer identifier is used to calculate the enhancing matching factor between two proteins.
- The *inhibitor identifier*, encoded as an integer between 0 and u_{size}. The inhibitor identifier is used to calculate the inhibiting matching factor between two proteins.
- The *type*, either *input*, *output*, or *regulator*. The type is a constant set by the user and is not evolved.

Each protein has a concentration, representing the use of this protein and proving state to the network similar to neurotransmitter concentrations in spiking neural networks (SNN). For *input* proteins, the concentration is given by the environment and is unaffected by other proteins. *output* protein concentrations are used to determine actions in the environment; these proteins do not affect others in the network. The bulk of the computation is performed by *regulatory* proteins, an internal protein whose concentration is influenced by other *input* and *regulatory* proteins.

The dynamics of the GRN are calculated as follows. First, the absolute affinity of a protein a with another protein b is given by the enhancing factor u^{+}_{ab} and the inhibiting u^{-}_{ab}:

$$u^{+}_{ij} = u_{size} - |enh_j - id_i| \; ; \; u^{-}_{ij} = u_{size} - |inh_j - id_i| \qquad (1)$$

where id_x is the identifier, enh_x is the enhancer identifier and inh_x is the inhibitor identifier of protein x. The maximum enhancing and inhibiting affinities between all protein pairs are determined and are used to calculate the relative affinity, which is here simply called the affinity:

$$A^{+}_{ij} = \beta(u^{+}_{ij} - u^{+}_{max}) \; ; \; A^{-}_{ij} = \beta(u^{-}_{ij} - u^{-}_{max}) \qquad (2)$$

β is one of two control parameters used in a GRN, both of which are described below. Variants of this equation used in this study are detailed in §§ 4.2.

These affinities are used to then calculate the enhancing and inhibiting influence of each protein, following

$$g_i = \frac{1}{N}\sum_{j}^{N} c_j e^{A^{+}_{ij}} \; ; \; h_i = \frac{1}{N}\sum_{j}^{N} c_j e^{A^{-}_{ij}} \qquad (3)$$

where g_i (resp. h_i) is the enhancing (resp. inhibiting) value for a protein i, N is the number of proteins in the network, c_j is the concentration of protein j. The use of an exponential in this function, as opposed to other operations, is examined in §§ 4.3

The final modification of protein i concentration is given by the following differential equation:

$$\frac{dc_i}{dt} = \frac{\delta(g_i - h_i)}{\Phi} \qquad (4)$$

where Φ is a function that normalizes the output and regulatory protein concentrations to sum to 1. The use of normalization is explored in §§ 4.4

β and δ are two constants that determine the speed of reaction of the regulatory network. The higher these values, the more sudden the transitions in the GRN. The lower they are, the smoother the transitions. For this paper, they are evolved as part of the GRN chromosome and are both kept within the range $[0.5, 2.0]$.

3.1 GRNEAT

Gene Regulatory Network Evolution Through Augmenting Topologies (GRNEAT) is a specialized Genetic Algorithm for GRN evolution proposed in [5] and based on Stanley's NeuroEvolution of Augmenting Topologies (NEAT) algorithm [13]. This algorithm has been shown to improve evolution performance on neural networks, complex pattern producing networks, and GRNs[12]. A major contribution is the design of a crossover method for network controllers, in which structure has a significant influence on overall network behavior. GRNEAT uses a similar crossover, and imports three key elements from NEAT:

- the initialization of the algorithm - small networks are generated that resemble a select subpopulation termed initial species leaders
- speciation, which limits competition and crossover to similar individuals. This both protects some new mutants from immediately competing with champion individuals and protects novel solutions by allowing them to optimize their structures before competing with the whole population
- an alignment crossover that compares individual genes before selection for a new individual

The distance metric in this paper for speciation and alignment crossover was

$$D_{prot}(i,j) = \frac{a|id_i - id_j| + b|enh_i - enh_j| + c|inh_i - inh_j|}{u_{size}} \qquad (5)$$

where $a = 0.75$, $b = 0.125$, and $c = 0.125$. Proteins were aligned in each GRN during comparison first based on *type* and secondly based on minimum $D_{prot}(A, B)$. The distance between GRNs was then calculated as

$$D(G_1, G_2) = \frac{D_{in} + D_{out} + D_{reg} + D_\beta + D_\delta}{max(N_1, N_2) + 2}$$
$$D_\beta = \frac{\beta_1 - \beta_2}{\beta_{max} - \beta_{min}}$$
$$D_\delta = \frac{\delta_1 - \delta_2}{\delta_{max} - \delta_{min}}$$

where N_i is the number of proteins in GRN G_i, and D_{type} is the sum of the difference of all aligned proteins of that *type*. For regulatory proteins, where N can differ between two GRNs, the distance for all non-aligned proteins was taken between the protein parameters and u_{size}. The use of alignment without replacement for the distance metric is novel in this work and differentiates it from [5]. This was found to improve results in preliminary trials but is not presented as a part of this study.

3.2 Evolution

Apart from the differences listed above, GRNEAT functions as a standard GA. The mutation operations available during this work were

- Modify (probability $p_{modify} = 0.25$)
- Add a new regulatory protein with random parameters, $p_{add} = 0.5$
- Delete $p_{delete} = 0.25$

When mutating a genome, a random mutation operation was selected with probability p_{select}. The delete mutation operation was not allowed when selected for input and output genes.

During the crossover mutation, aligned proteins are randomly selected from either parent with probability $p_{cross} = 0.5$. If the parent genomes are of different lengths, the unaligned regulatory proteins from the longer parent genome are appended to the child genome with probability $p_{append} = 0.5$.

4 DYNAMICS

In this work, we focused on specific improvements to the GRN that either vary in the literature or could be used to simplify GRN implementation. These modifications impact the GRN encoding e, affinity metric a, the influence function f, the and the normalization step n. For each of these modifications, we evaluate the potential fitness contribution in § 6.

Equation 4 can be generalized with f, a, and n as follows:

$$\frac{dc_i}{dt} = n\left(\frac{\delta}{N} \sum_j^N c_j(f(a^+(i,j)) - f(a^-(i,j))\right) \quad (6)$$

with the encoding e changing the evolution dynamics, the parameter and the possible range of u_{ij}. e, a, f, and n are all described below.

4.1 Encoding

In early versions of GRNs, and in some modern implementations, proteins were encoded in binary format. The affinity between two proteins was defined by the number of bits in common, and mutation and crossover operations happened at a binary level. [1] For this reason, integer encodings of GRNs often use 32 as the maximum *identifier* size, u_{size}, as proteins were 32 bits long. Here we strive to simplify this model by proposing real values between $[0, 1]$ for the protein tags. The mutation operation is altered as a result, as the distance change from a mutation can be smaller than an integer step size. The aligned crossover operation is also affected, as the distance between proteins operates on real values. Lastly, u_{size} is set to 1.0 in this encoding, which affects the affinity metric.

4.2 Affinity metric

In [1] and in many works since, the affinity metric uses the maximum affinity metric, u_{max}, as a scaling factor. This is the maximum of the relative affinity metric, $u_{size} - |enh_j - id_i|$, (resp *inh*) across all (i, j). However, for a reasonably large network, in which two protein tags will become arbitrarily close, u_{max} will approach u_{size}. For this reason, we propose the following novel affinity metric, which reduces complexity by removing the maximization factor:

$$A^+_{ij} = -\frac{\beta|enh_j - id_i|}{u_{size}} \; ; \; A^-_{ij} = -\frac{\beta|inh_j - id_i|}{u_{size}} \quad (7)$$

In comparison with the original equation, it was noted that on some implementations, only u_{ij} was multiplied by β. This results in the following equation:

$$\begin{aligned} A^+_{ij} &= \beta(u_{size} - |enh_j - id_i|) - u^+_{max} \\ A^-_{ij} &= \beta(u_{size} - |inh_j - id_i|) - u^-_{max} \end{aligned} \quad (8)$$

Lastly, the original affinity metric is evaluated:

$$\begin{aligned} A^+_{ij} &= \beta(u_{size} - |enh_j - id_i| - u^+_{max}) \\ A^-_{ij} &= \beta(u_{size} - |inh_j - id_i| - u^-_{max}) \end{aligned} \quad (9)$$

4.3 Influence function

In [1], [7], and many others, an exponential function of the affinity is used to determine the influence of one protein onto an other. This is the first influence function we evaluate:

$$f(A_{ij}) = e^{A_{ij}} \quad (10)$$

In [10], a hyperbolic tangent function is used. The constants of this implementation were modified to provide the same results as Equation 10 at $A_{ij} = 0$:

$$f(A_{ij}) = tanh(A_{ij}) + 1 \quad (11)$$

[9] uses an inverse exponential, and also decreases the protein concentration by this influence instead of increasing. As such, the inverse exponential in this work is modified to match the same relationship as the other metrics, and the constants are again modified to provide the same results as Equation 10 at $A_{ij} = 0$:

$$f(A_{ij}) = \frac{2}{1 + e^{-A_{ij}}} \quad (12)$$

4.4 Normalization

An important component of GRN dynamics is the normalization of protein concentrations at each step, such that the output and regulatory protein concentrations sum to 1. This makes the output

Table 1: GRN modifications evaluated and their corresponding labels

$s = 0$	integer encoding scheme
$s = 1$	real encoding scheme
$a = o$	Equation 7, simply using u_{ij}
$a = 1$	Equation 8, with u_{max} outside β's influence
$a = 2$	Equation 9, the original from [1]
$f = 0$	Equation 10, e
$f = 1$	Equation 11, $tanh$
$f = 2$	Equation 12, inverse e
$n = 0$	normalization of concentrations by their sum
$n = 1$	constraining all concentrations to $[0.0, 1.0]$

layer of the GRN function similarly to the softmax layer of modern ANNs, and is often useful in problem implementation. However, it can also be a difficult concept to grasp when understanding GRNs, increases implementation and computation complexity, and requires a knowledge of common GRN inputs and outputs for good problem design. For example, in the experiments in § 5, some of the control problem outputs are designed with normalization in mind by forcing an action only when one output concentration exceeds another, favoring the periodic dynamics resultant from this normalization step.

In this work, we propose the simple use of boundaries [0, 1] for protein concentrations as an alternative to normalization. We refer to this method as capping.

4.5 Simplification

In review, the modifications proposed, and the variables used to denote them, are as listed in Table 1. We propose the model corresponding to $s = 1$, $a = 0$, $f = 0$, $n = 1$ as a simplified GRN The dynamics of this GRN implementation are, according to its modifications:

$$\frac{dc_i}{dt} = \frac{\delta}{N} \sum_{j}^{N} c_j (e^{-\beta|enh_j - id_i|} - e^{-\beta|inh_j - id_i|})$$

$$c_{i,t+1} = max(min(c_{i,t} + \frac{dc_i}{dt}, 1.0), 0.0)$$

This GRN formula reduces complexity by using the simplest affinity metric, $a = 0$, which does not include determining u_{max}. It is a real encoding, which removes the determination of the u_{size} parameter. Finally, it uses a min-max step, $n = 1$, instead of normalization of protein concentrations, which is computationally complex and another implementation step. We evaluate the performance of this model to determine if it is equally capable, as well as evaluating all modifications independently.

5 EXPERIMENT

To evaluate the impact of each modification mentioned above, we have used standard problems from a broad spread of the literature. More specifically, GRNs are often used as real-time controllers, as they are in the following signal processing, robot control, and game problems.

Table 2: GRNEAT parameters used in all experiments

initial population	500
generations	300
tournament size	3
minimum species size	15
number of elites per species	1
speciation threshold	0.3
maximum speciation threshold	0.8
minimum speciation threshold	0.01
mutation rate	0.75
crossover rate	0.25

For each problem, all 36 GRN modification combinations (2 encoding schemes, 3 affinity equations, 3 influence functions and 2 normalization methods) were evaluated over 40 runs. The parameters of the evolutions can be found in Table 2.

The code used for all following experiments, including the problems and the details of their parameters, the GRN and GRNEAT, are available in C++ on GitHub[1]. Video of the best performing GRN on the Ship Escape problem is also available.

5.1 Doubling input frequency

In this first problem, from [9], a sinusoidal signal of frequency F_i is fed to the GRN by varying the concentration of its only input protein. The goal of the GRN is to make the concentration of its output protein to vary at twice the input frequency, i.e $F_o = 2 * F_i$, with F_o being the variation frequency of the output protein. We used the same fitness function as in § 5, being the sum of the absolute distances between the desired and obtained signal at each time step, divided by the absolute distance between F_o and F_i. The input signal is divided into three sequences of equal length (1000 time steps) and at frequencies equals to 125Hz, 500Hz and finally 0Hz (flat signal). The proteins concentrations are reset between each sequence.

5.2 Low pass frequency filter

In another classic signal treatment problem from [9], the GRN must act as a low pass filter, meaning it must strip the input signal of any frequency greater than the cut-off frequency $F_c = 50$Hz. Here, the fitness is the average squared distance between the output signal (scaled by a constant factor $C = 5$ in order to not penalize the normalized concentrations implementations).

The input signal is divided into three sequences of equal length (1000 time steps):

- A signal composed of 3 combined subsignals: one at 7Hz with an amplitude of 0.7, another at 250Hz with an amplitude of 0.2 and the last one at 1250Hz and an amplitude of 0.1. The desired output signal should have a frequency of 7Hz with an amplitude of 0.7
- A signal composed of 4 combined subsignals: one at 17Hz with an amplitude of 0.4, another at 350Hz with an amplitude of 0.2n, a third one with a frequency 1100Hz and an amplitude of 0.2 and the last one at 2000Hz and an

[1] https://github.com/jdisset/grnbenchmarks/

amplitude of 0.1. The desired output signal should have a frequency of 17Hz with an amplitude of 0.4

- A flat "zero" signal, which should be exactly reproduced at the output.

The GRN proteins concentrations aren't reset between each sequences.

5.3 Coverage control

This is a classic coverage problem where the GRN controls a robot in a 2D grid. It has 8 inputs: the number of obstacles on the next 3 grid cells in each four directions (north, south, east, west), one protein per direction, and the number of unexplored cells in each directions. It has 4 output proteins (one for each direction), the protein with the highest concentrations deciding the direction in which the robot will move at the next time step. Each GRN runs for 200 steps on 3 different 10 by 10 maps with 20 obstacles and the fitness is the average of the discovered portions of the maps.

5.4 Flappy Bird

In order to challenge the capabilities of a GRN as a game AI controller, we implemented a version of the famous small game Flappy Bird. In this game, a small bird progresses through an horizontal world, bounded by a ceiling and a floor. It must pass through gates whose positions and aperture height are randomly generated. The only control the player has over the bird is the timing of its wings' flaps, which provide upward thrust. The horizontal speed increases over time. Here, we defined three input proteins for our GRN, whose concentrations respectively corresponds to:

- the bird's height, normalized by the height of the world
- the next gate's position, normalized by the length of the screen
- the next gate's aperture height, normalized by the height of the world

The flap's timings are controlled by concentrations c_{o1} and c_{o2} of its two output proteins. The bird flaps its wings each time $c_{o1} > c_{o2}$. The fitness is equal to the average horizontal distance at which the bird first hit a gate, the floor, or the ceiling.

5.5 Ship Escape

The last problem requires the most inputs and outputs. Here, the GRN must learn to drive a ship in a vertical world bounded by walls and filled with randomly placed obstacles. The goal is to drive the ship as far as possible without hitting anything, with the added dificulty of gates slowly closing ahead of the ship. The distance between two gates increases after each passed one, and the next gate starts to close as soon as the previous one has been passed through. This puts pressure on the ship to go accelerate while still avoiding the obstacles. For this problem, we used 13 inputs: 11 of them represented laser beams casted by the ship in 11 evenly distributed directions, from $\frac{\pi}{2}$ to $-\frac{\pi}{2}$ relatively to the ship direction, each beam directly setting the concentration of an input protein C_i as equal to $\frac{D_i}{H}$, i.e the distance betweed the ship and the nearest obstacle in the direction of the beam, normalized by a maximum distance H. The remaining two inputs are indications of the ship's orientation, being set respectively as $sin(\theta)$ and $cos(\theta)$, with θ the current oriented angle of the ship.

The ship also needs to have output that allow it to control its direction and its propulsion. To do so, we add 3 pairs of output proteins: one pair that will allow the ship to turn left when the concentration of the first protein of the pair goes above the concentration of the other, one pair to turn right using the same principle, and one pair to turn the engine on using again the same principle.

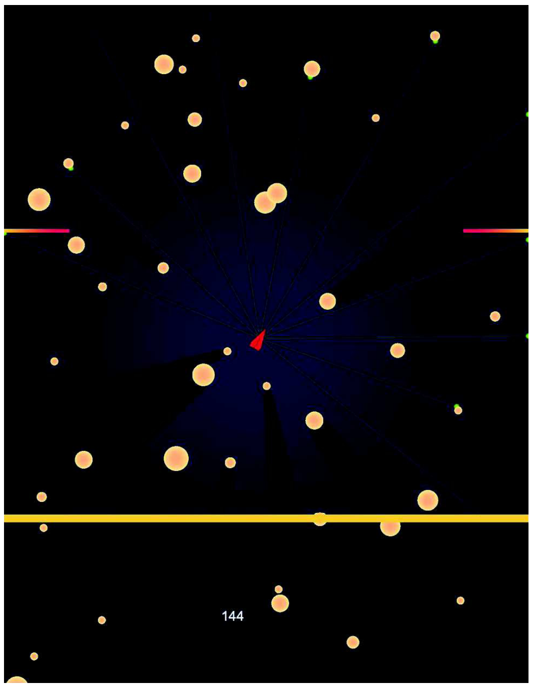

Figure 1: The Ship Escape problem. The lines from the ship show the laser beams used for input

6 RESULTS

To independently compare the impact of each modification, we first compare pairs of implementations with only one modification, such as $e = 0, a = 0, f = 0, n = 1$ to $e = 0, a = 0, f = 2, n = 1$. For each pair, the 40 runs were used to fit two normal distributions using maximum likelihood estimation, $N(\mu_1, \sigma_1)$ and $N(\mu_2, \sigma_2)$. The probability that values from one distribution are greater than the values from the second distribution is used to compute a Competitive Probability Score (CPS). The CPS of a modification is the summed probability difference for each modification pair. To determine this, first the competitive probabilities for implementations with all but one modification in common are summed, here shown for f:

$$P_f[i,j] = \frac{1}{\eta}\sum_i\sum_j 1 - \Phi(\frac{\mu_2 - \mu_1}{\sqrt{\sigma_1 + \sigma_2}})$$
$$if s_i == s_j, a_i == a_j, n_i == n_j$$

	Doubling			Lowpass			Coverage			Flappy Bird			Ship Escape		
$s = 0$	0.306	0.327	0.306	**0.516**	0.492	**0.504**	0.413	0.425	0.419	0.451	0.421	0.446	0.418	0.421	0.426
$s = 1$	**0.693**	**0.672**	**0.693**	0.483	**0.507**	0.496	**0.586**	**0.575**	**0.581**	**0.548**	**0.578**	**0.553**	**0.581**	**0.578**	**0.573**
$a = 0$	0.421	0.411	0.423	**0.522**	**0.517**	**0.518**	0.463	0.471	0.474	0.500	0.500	0.498	**0.513**	**0.511**	**0.512**
$a = 1$	0.528	0.541	0.532	0.492	0.508	0.505	**0.546**	**0.530**	**0.534**	**0.533**	**0.522**	**0.529**	0.494	0.501	0.498
$a = 2$	**0.550**	**0.546**	**0.544**	0.485	0.474	0.476	0.490	0.497	0.491	0.465	0.476	0.472	0.491	0.486	0.488
$f = 0$	0.434	0.445	0.443	0.502	**0.515**	**0.508**	**0.538**	**0.518**	**0.524**	0.500	0.496	0.502	0.484	0.488	0.484
$f = 1$	**0.548**	**0.542**	**0.544**	**0.503**	0.496	0.499	0.493	0.501	0.499	**0.508**	**0.514**	**0.511**	0.494	0.492	0.493
$f = 2$	0.517	0.512	0.512	0.494	0.487	0.492	0.468	0.480	0.475	0.491	0.489	0.486	**0.521**	**0.518**	**0.521**
$n = 0$	**0.563**	**0.564**	**0.563**	0.498	0.495	0.497	**0.590**	**0.592**	**0.591**	**0.504**	**0.519**	**0.513**	**0.521**	**0.516**	**0.520**
$n = 1$	0.436	0.435	0.436	**0.501**	**0.505**	**0.502**	0.410	0.407	0.408	0.495	0.480	0.486	0.478	0.483	0.479

Table 3: Independent modification CPS over all 5 problems, on an early generation g, the final generation, and the average over all generations. The best modification for each type is highlighted.

	s	a	f	n
0.738265	1	1	0	0
0.726697	1	0	1	0
0.710317	1	0	0	0
0.703542	1	1	2	0
0.697183	1	2	1	0

Table 4: The top five implementations based on CPS

	s	a	f	n
doubling	9.176e-38	1.069e-08	1.215e-03	1.597e-07
lowpass	3.044-16	0.012	0.725	0.544
coverage	2.572e-18	0.046	0.080	3.645e-17
flappy	8.808e-20	0.089	0.233	0.062
ship	1.196e-16	0.175	0.715	0.150

Table 5: One-way ANOVA p values between the different implementations for each method on the 5 problems

η is the number of implementations in the sum, and Φ is the cumulative distribution function of the normal distribution $N(0, 1)$, making $P_f[i, j]$ the average probability that modification $f = i$ is greater than $f = j$. The CPS is then simply the sum:

$$CPS(f = i) = \sum_j P_f[i, j] \qquad (13)$$

The CPS for each modification is presented in Table 3, with the most rewarding modification of each type highlighted. The CPS is presented at generation $g = 100$, $g = 300$, and the normalized sum over all g, to evaluate whether some modifications lead to better early convergence or overall performance.

The global best GRN implementation was determined by the same process as the CPS, only over entire implementations. Each of the 36 implementations were compared and a normal distribution was fit to the results from their last generation. The probability of each one exceeding the other was summed and the implementation with the highest average probability over all other implementations was chosen. The top 5 implementations using this metric are shown in Table 4.

The clear trends from Table 3 and Table 4 are the advantage of the real encoding and the necessity of the normalization step. The best implementation, $s = 1, a = 1, f = 0, n = 0$ is surprising to us given the use of $a = 1$, which uses a non-scaled u_{max}. We believe this result to be sensitive to the β range, and that a clear affinity metric is not determined by the results. Similarly, it seems that $f = 0$ is the best influence function, but this is not as conclusive as the advantages of $s = 1$ and $n = 0$.

Analysis of variance (ANOVA) tests were also conducted for each problem. Groups were constructed for each implementation within a method, such as a group corresponding to $s = 0$ and another to $s = 1$, and the variance of fit distributions to the groups were evaluated. The p values from this analysis are presented in Table 5. The findings of this analysis show a clear difference across problems in s and low p values for most problems on a and n. These results demonstrate that the implementations can be significantly different depending on the problem.

While the simplified GRN proposed, $s = 1, a = 0, f = 0, n = 1$, was overall the seventh best implementation, the second best and third best implementations use $a = 0$, the simplest affinity metric, and all of the top implementations use $s = 1$, which simplifies encoding and reduces parameters.

In the signal processing benchmark problems, the top implementations were not significantly different and all performed similarly to results found in [5]. While some implementations fared very poorly, we found these results not independently conclusive for determining implementation fitness. While the low pass experiment has the only best implementation using $s = 0$, it was not a significant advantage over the other implementations on this rather simple problem.

The implementations used here show improvement on the coverage control problem over [5], but performance didn't vary significantly over different implementations. On this problem, the best implementation is also the global best.

The results on the Flappy Bird problem are impressive for their variety and for the best performance. A human user scored an average of 20.349 over 20 trials, and we find it difficult to believe that a human player could achieve the same scores as the best GRN.

As in the Flappy Bird problem, the GRN performance on the Ship Escape problem rivals or outperforms human capability.

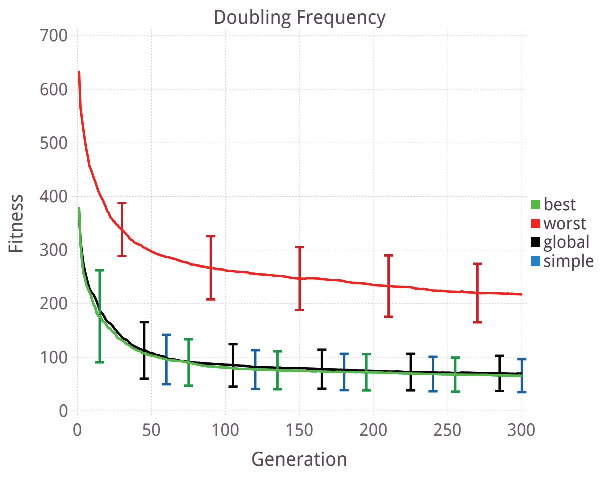

Figure 2: The best implementation $s = 1, a = 0, f = 0, n = 1$ and the worst implementation $s = 0, a = 0, f = 2, n = 1$ of the doubling frequency problem compared to the global best and the proposed simplified GRN.

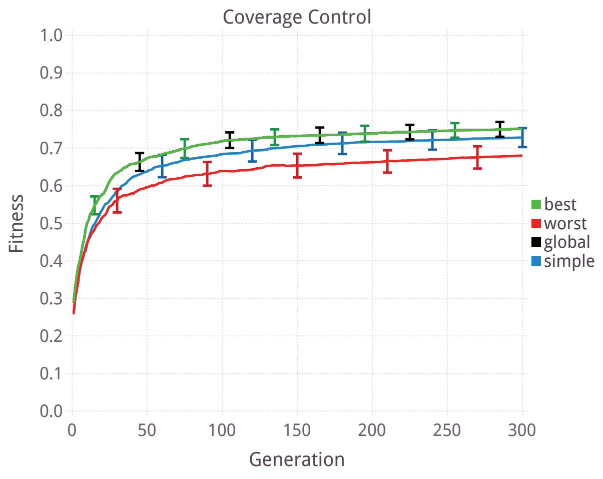

Figure 4: Comparison with the best implementation $s = 1, a = 1, f = 0, n = 1$ and the worst implementation $s = 0, a = 0, f = 0, n = 1$ of the coverage control problem

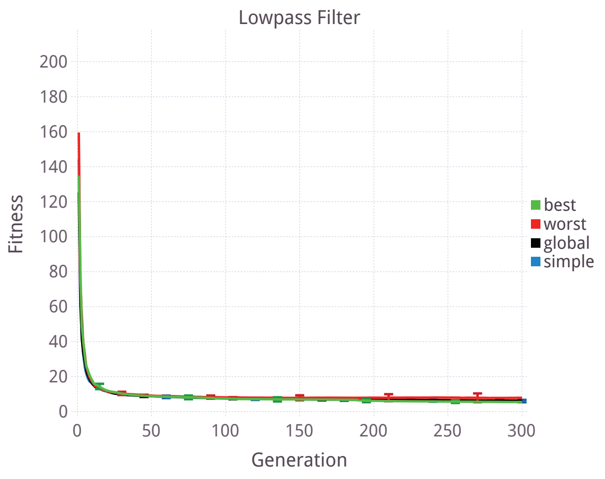

Figure 3: Comparison with the best implementation $s = 0, a = 1, f = 0, n = 1$ and the worst implementation $s = 0, a = 0, f = 1, n = 0$ of the low pass filter problem

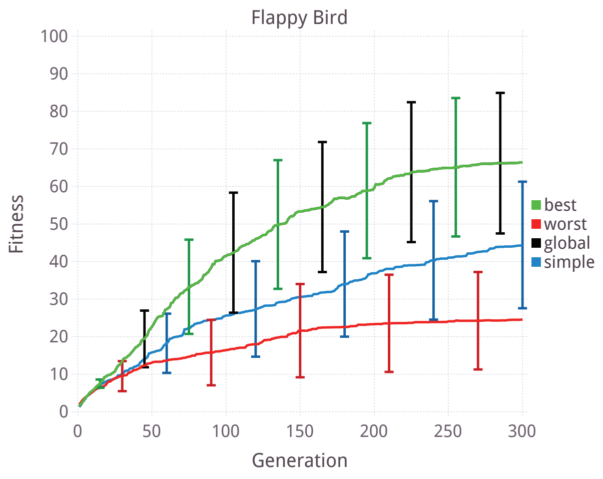

Figure 5: Comparison with the best implementation $s = 1, a = 0, f = 1, n = 0$ and the worst implementation $s = 0, a = 2, f = 0, n = 1$ of the Flappy Bird problem

7 DISCUSSION AND CONCLUSION

In this paper we have reviewed several variations of GRNs and tried new combinations of various literature implementations. The clear advantages found are the use of real numbers for proteins coordinates and the normalization of proteins concentrations. However, we note that the normalization step may have received a bias in this work due to the problem output definitions. One of the complexities introduced by the normalization step is the need for output protein design compatible with normalization. The design of the Flappy Bird and Ship Escape outputs, in particular, use a common mechanism of comparison of two output proteins to determine action, which is done with normalization in mind. It is possible that $n = 1$, the capping method instead of normalization, would have fared better with different problem design.

Other than these two important points, it is difficult to draw significant conclusions on an "overall best" GRN implementation, as different GRNs dominated each different problem. However, an overall simplification can be proposed with the use of the real encoding scheme, and we encourage use of the simpler $a = 0$ affinity metric scheme as well, as it achieves similar results to the best across the used benchmarks.

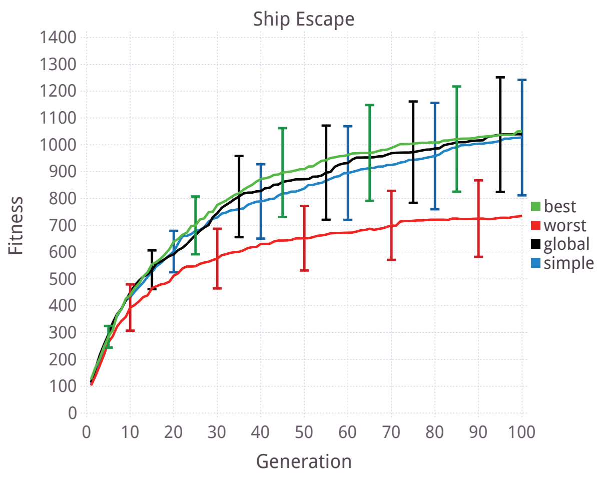

Figure 6: Comparison with the best implementation $s = 1, a = 2, f = 1, n = 0$ and the worst implementation $s = 0, a = 2, f = 0, n = 0$ of the ship escape problem

It is also important to note that this paper focused only on different GRN implementations in an attempt to see more clearly through the variations. While a comparison of GRNs on these benchmark problems to other controllers, such as Genetic Programming, ANNs, or HyperNEAT[12], is of interest, it was not in the scope of this work. We welcome the use of these benchmarks and results as a basis of comparison.

Overall, the experiments here demonstrated, as has been done before, the quality performance of GRNs as controllers, but it also highlight the difficulties inherent to their use. On complex problems especially, the observed high standard deviations for the best fitnesses highlight the troubles one can stumble upon trying to reliably and efficiently evolve good GRNs. This seems to be especially the case with problems where a large number of inputs and/or outputs are needed, such as the Ship Escape problem.

Given the performance of the global best GRN implementation in this paper over the five problems proposed, an interesting next step is to use the same GRN on multiple problems. The top implementations all show consistent performance across this set of problems, so we are interested in pursuing an evolution method that matches this performance using a single GRN. Furthermore, we could expand this to use benchmark suites that are gaining popularity in the deep learning field.

Lastly, while attempts to greatly simplify the GRN were somewhat thwarted in this work, there is still more to be done to decrease the usage complexity of GRNs, specifically in their interface design. With a real valued encoding, a positional interface scheme could be used in embedded controller scenarios, where physical interface location matches GRN input and output design. The exploration of this scheme and its potential for simplification is left to future discussion.

Acknowledgments

This work is supported by ANR-11-LABX-0040-CIMI, within programme ANR-11-IDEX-0002-02. This work was performed using HPC resources from CALMIP (Grant P16043).

REFERENCES

[1] Wolfgang Banzhaf. 2003. Artificial regulatory networks and genetic programming. In *Genetic programming theory and practice*. Springer, 43–61.

[2] Arturo Chavoya and Yves Duthen. 2008. A cell pattern generation model based on an extended artificial regulatory network. *BioSystems* 94, 1-2 (2008), 95–101. DOI:http://dx.doi.org/10.1016/j.biosystems.2008.05.015

[3] Sylvain Cussat-Blanc, Nicolas Bredeche, Hervé Luga, Yves Duthen, and Marc Schoenauer. 2011. Artificial Gene Regulatory Network and Spatial Computation: A Case Study. *European Conference on Artificial Life* (2011).

[4] Sylvain Cussat-Blanc and Kyle Harrington. 2015. Genetically-regulated Neuromodulation Facilitates Multi-Task Reinforcement Learning. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. ACM Press, New York, New York, USA, 551–558. DOI:http://dx.doi.org/10.1145/2739480.2754730

[5] Sylvain Cussat-Blanc, Kyle Harrington, and Jordan Pollack. 2015. Gene Regulatory Network Evolution Through Augmenting Topologies. *IEEE Transactions on Evolutionary Computation* 19, 6 (dec 2015), 823–837. DOI:http://dx.doi.org/10.1109/TEVC.2015.2396199

[6] S Cussat-Blanc, H Luga, and Yves Duthen. 2008. From single cell to simple creature morphology and metabolism. *Artificial Life XI* (2008), 134–141. http://www.cs.bham.ac.uk/{~}wb

[7] P. Dwight Kuo, Wolfgang Banzhaf, and André Leier. 2006. Network topology and the evolution of dynamics in an artificial genetic regulatory network model created by whole genome duplication and divergence. *BioSystems* 85, 3 (2006), 177–200. DOI:http://dx.doi.org/10.1016/j.biosystems.2006.01.004

[8] Michał Joachimczak and Borys Wrobel. 2010. Evolving Gene Regulatory Networks for Real Time Control of Foraging Behaviours. *Artificial Life XII. Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems* (2010), 348–355.

[9] M.a Joachimczak and B.a b Wróbel. 2010. Processing signals with evolving artificial gene regulatory networks. *Artificial Life XII: Proceedings of the 12th International Conference on the Synthesis and Simulation of Living Systems, ALIFE 2010* (2010), 203–210. http://www.scopus.com/inward/record.url?eid=2-s2.0-84874698466

[10] Johannes Knabe, Chrystpoher Nehaniv, Maria Schilstra, and Tom Quick. 2006. Evolving Biological Clocks using Genetic Regulatory Networks. *Artificial Life X : Proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems* Alife 10 (2006), 15–21. c:/Daniel/Work/Library/workLibrary.Data/PDF/2088839662/Knabe2006.pdf

[11] Miguel Nicolau, Marc Schoenauer, and Wolfgang Banzhaf. 2010. Evolving genes to balance a pole. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6021 LNCS (2010), 196–207. DOI:http://dx.doi.org/10.1007/978-3-642-12148-7-17 arXiv:1005.2815

[12] Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. 2009. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life* 15, 2 (2009), 185–212.

[13] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127. DOI:http://dx.doi.org/10.1162/106365602320169811

[14] M.A. Trefzer, T. Kuyucu, Julian F Miller, and A.M. Tyrrell. 2013. On the Advantages of Variable Length GRNs for the Evolution of Multicellular Developmental Systems. *IEEE Transactions on Evolutionary Computation* 17, 1 (2013), 100–121. DOI:http://dx.doi.org/10.1109/TEVC.2012.2185848

[15] Dennis Wilson, Emmanuel Awa, Sylvain Cussat-Blanc, Kalyan Veeramachaneni, and Una-May O'Reilly. 2013. On learning to generate wind farm layouts. *Fifteenth annual conference on Genetic and evolutionary computation conference* (2013), 767–774. DOI:http://dx.doi.org/10.1145/2463372.2463462

[16] Borys Wróbel and Ahmed Abdelmotaleb. 2012. Evolving Spiking Neural Networks in the GReaNs (Gene Regulatory evolving artificial Networks) Plaftorm. *EvoNet2012: Evolving Networks, from Systems/Synthetic Biology to Computational Neuroscience Workshop at Artificial Life XIII* (2012), 19–22.

[17] Borys Wrobel, Ahmed Abdelmotaleb, and Michał Joachimczak. 2014. Evolving networks processing signals with a mixed paradigm, inspired by gene regulatory networks and spiking neurons. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST* 134 (2014), 135–149. DOI:http://dx.doi.org/10.1007/978-3-319-06944-9_10