Extracting Key Gene Regulatory Dynamics for the Direct Control of Mechanical Systems

Jean Krohn and Denise Gorse

Department of Computer Science, UCL (University College London),
Gower Street, London WC1E 6BT, UK
{j.krohn,d.gorse}@cs.ucl.ac.uk

Abstract. Evolution produces gene regulatory networks (GRNs) able to control cells. With this inspiration we evolve artificial GRN (AGRN) genomes for the reinforcement learning control of mechanical systems with unknown dynamics, a problem domain similar in its sparse feedback to that of controlling a biological cell. From the fractal GRN (FGRN), a successful but complex GRN model, we obtain the Input-Merge-Regulate-Output (IMRO) abstraction for GRN-based controllers, in which the FGRN's complex fractal operations are replaced by simpler ones. Computational experiments on reinforcement learning problems show significant improvements from the use of this simplified approach. We also present the first evolutionary solution to a hardened version of the acrobot problem, which previous evolutionary methods have failed on.

Keywords: Gene regulatory network, IMRO, FGRN, genetic algorithm, ALPS, control, reinforcement learning, pole balancing, acrobot.

1 Introduction

Gene regulatory networks (GRNs) act as controllers in situations as different as single cell bacteria and multi-cell organisms, with orders of magnitude of variation in size. GRNs are a product of evolution, optimised for controlling their host cell in a myriad ways, depending on context (single-cell vs multi-cell organism, during development, etc).

However it is generally desirable to avoid unnecessary complexity when developing systems inspired by nature and hence we will seek here to extract the key dynamics of the FGRN model, currently the most successful AGRN model for control, to improve its control capabilities. In parallel with nature we use artificial evolution, in the form of a genetic algorithm (GA), to evolve AGRNs for the control of mechanical systems.

This paper focuses on mechanical systems reinforcement learning problems, in which the system's dynamics are completely unknown and the reinforcement feedback is limited. Unknown dynamics is important for real world applications (for example coal furnace combustion control [5]), and in addition by minimising domain knowledge we are ensuring the wide applicability of our method. This work additionally represents the first solution of the double pole and acrobot problems with an AGRN system.

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part I, LNCS 7491, pp. 468–477, 2012.

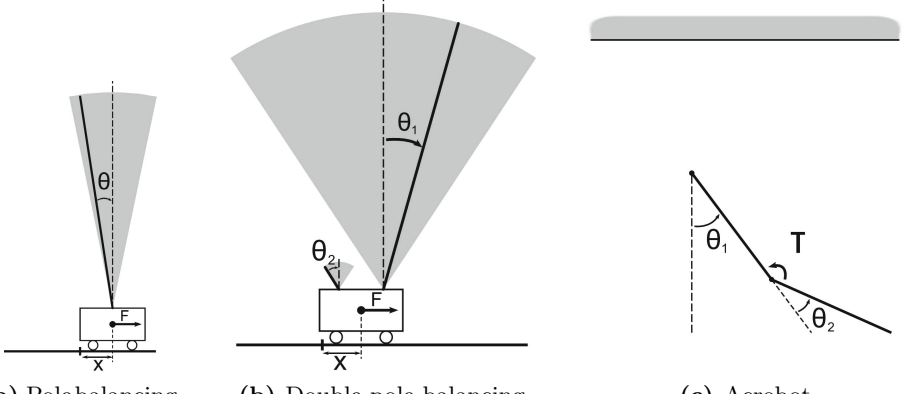
2 Related Work on Artificial GRNs

Surprisingly, given the role of natural GRNs in biological control, AGRN applications were initially focused only on development. But recently AGRN models have known increased use for control: Nicolau et al[12], and Lopes and Costa[11], have applied GRNs based on a binary genome to single pole balancing, while Joachimczak and Wróbel[8] used a GRN based on a linear genome for evolving simple foraging behaviours.

The FGRN system, introduced by Bentley[2], is a complex evolutionary model of a GRN in which proteins are bitmaps generated from the Mandelbrot set fractal. The FGRN was originally devised as a developmental system with applications such as pattern generation, or an algorithm producing increasingly precise estimations of π[9], though it also had clear potential for control purposes. There have been a number of previous FGRN control applications, evolving behaviours such as wall-following[1], grid-world box-pushing[17], and robotic locomotion[16]. In particular Krohn and Gorse[10] have applied the FGRN system to multiple versions of the single pole balancing problem, a starting point for the more ambitious work to be presented here.

3 Problem Domain: Reinforcement Learning

Reinforcement learning consists of discovering what actions to take for any given environmental state in order to maximise a scalar reward[15]. In this paper the focus will be on problems in which the full state of the environment is given, but in which, as discussed above, the environmental dynamics are unknown. Figure 1 displays the problems considered in this work.



(a) Pole balancing **(b)** Double pole balancing **(c)** Acrobot

Fig. 1. Reinforcement learning problems considered in this work

3.1 Pole Balancing

Pole balancing is a well-known, well-studied control problem that has been used as a benchmark for the design and test of many controllers[6]. Consequently, standard equations of motion and constants have arisen; these will be used in the current work and are described in ref [6].

The single pole version, shown in Figure 1a, is usually (and here) defined to be the problem of keeping the angular position θ of the 1.0m tall hinged pole within 12° of vertical, and the distance x of the cart on which it is mounted within 2.4m of the centre of the track, using only 'bang-bang' control (a force F of $\pm$10N being applied to the cart at each time step). The system state information given to the controller consists of x, $\dot{x}$, θ, $\dot{\theta}$.

The double pole version, shown in Figure 1b, consists of simultaneously balancing two poles of different size, with different starting angles, on the cart. The poles are respectively 1.0m and 0.1m tall, and the acceptable range for pole angles θ_1 and θ_2 is here within 36° of vertical. In this case the system state is composed of x, $\dot{x}$, θ_1, $\dot{\theta}_1$, θ_2, $\dot{\theta}_2$. Most other studies involving double pole balancing use as integration method two-step fourth order Runge-Kutta, and for consistency this will be used throughout the current work.

Gomez et al. have produced an extensive comparison of the performance of machine learning methods on the single and double pole balancing problems with or without velocities included in the inputs[6]. Controllers developed using the pole balancing problem have also been used in a variety of real-world control applications[5].

3.2 Acrobot

The acrobot, shown in Figure 1c, is a two-link underactuated robot[13]; it is roughly analogous to a gymnast hanging from a bar and only able to act by bending at the hips. The acrobot has been extensively studied both as a control and machine learning problem.

Unlike the pole balancing problem the acrobot problem definition varies significantly from one study to the next. However acrobot goals can be put into two broad categories: swing-up and handstand. Swing-up consists of generating actions such that the acrobot's tip (the gymnast's feet) reaches a one link height above the bar in the shortest possible amount of simulated time. Handstand is the harder task of swinging up the acrobot and then keeping both links vertically balanced; *all* solutions to the acrobot handstand problem have so far included pre-existing knowledge of the problem, e.g. the equations of motion, the desired energy level of the goal position, or the coordinates of the target position[3]. Solutions to the swing-up problem have frequently also involved pre-existing domain knowledge, though Sutton[14] successfully applied a combination of SARSA with coarse input coding to the 5Hz swing up problem, with bang-zero-bang.

More recently, and most significantly for the current work, da Motta Salles Barreto and Anderson[4] have introduced a harder version of the acrobot swing-up problem by multiplying by four the frequency of control actions (using a 20Hz

rather than 5Hz control frequency), reporting successful results with a SARSA-based method and a policy iteration algorithm but being unable to obtain any viable solution using several evolutionary methods. In contrast this paper will show that by extracting key operational features from the FGRN model, and by using for inputs the continuous state representation $\sin\theta_1$, $\cos\theta_1$, $\sin\theta_2$, $\cos\theta_2$, $\dot{\theta_1}$, and $\dot{\theta_2}$, it is indeed possible to address this more challenging version of the problem using an evolutionary method.

4 System Description

4.1 Input-Merge-Regulate-Output (IMRO) System

The IMRO system is an abstraction of the GRN model that underpins the FGRN. An IMRO genome is a set of genes with one of three possible types (input, regulatory, or output). An IMRO controller is the combination of a genome and a *merging* module which, similarly to a biological cell, provides the environment for the 'execution' of the genome. The merging module takes in proteins and merges them into a cell state that changes with the addition of new proteins. The cell state is an array of real values of length N.

An *input* gene takes in a scalar input and produces a protein output; it is equivalent to a combination of the FGRN's environmental and receptor genes. Both *regulatory* and *output* genes take in the cell state, outputting proteins in the case of the former, and a scalar in the case of the latter. The outputs of the genes are functions only of their latest input, whereas the merging module stores past proteins until they have decayed by the mechanism to be described below.

One control iteration of the IMRO controller (in which the controller receives input and gives output) consists of the following steps: (i) the existing proteins are decayed; (ii) for each scalar input a corresponding input protein is generated and added to the existing proteins in the merging module; (iii) these proteins are combined into a new cell state by the merging module; (iv) the cell state determines the activation of the regulatory genes, which output proteins to the merging module, and also the activation of the output genes, which produce the controller's scalar outputs.

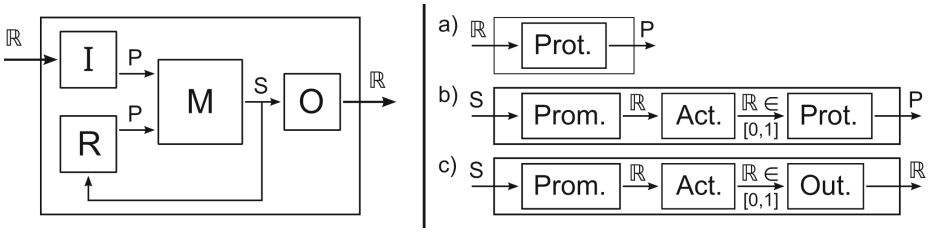


Fig. 2. The IMRO system. Left, the data flow of a controller. Right, the internal details of the a) **I**nput, b) **R**egulatory, and c) **O**utput genes. Key: P = protein, S = cell state, ℝ = real scalar

Proteins and Cell State. A protein is composed of an array of integers L of length N, a lifespan τ and a real value v. L is an array of levels, determining how prominently the protein will feature in the cell state; τ is the protein's time to live; and v determines how the protein will influence, through the cell state, the activation of regulatory and output genes. Proteins are decayed by decreasing τ by one; when $\tau = 0$ the protein is deleted.

The cell state array is generated by taking, for each i in N, the value v of the protein with the highest level L_i for that index (or the average of the v values, if several proteins have the same maximum L_i); if for the index i there is no existing protein value L_i superior to a fixed threshold set to 0, the corresponding value in the cell state is set to 0. This allows the evolution of proteins which only influence part of the cell state.

4.2 Gene Components

As detailed in Figure 2, the genes are composed of combinations of four components: promoter, activation, protein-output, and scalar-output. All component parameters are subject to evolution.

Gene Promoter. The role of a natural gene's promoter section is to regulate the activation of the gene based on the presence/absence of certain proteins or combinations of proteins. The IMRO gene promoter accomplishes this regulatory role by masking away part of the merged protein cell state, and then producing a matching score that is used further on to determine the activation of the gene.

In detail, the IMRO promoter consists of a pair of evolvable arrays of the same size N as the cell state: a boolean vector M acting as a mask, and a real vector W providing weights for the corresponding values in the cell state. Formally, the matching score $m_{i,t}$ of the promoter of gene i at time step t of a given simulation (e.g. a pole balancing run) is $m_{i,t} = \sum_{j=1}^{N} M_{i,j} W_{i,j} S_{j,t}$, where $M_{i,j} \in \{0, 1\}$ is the jth element of the promoter mask vector of gene i; $W_{i,j} \in \mathbb{R}$ is the jth element of the promoter weight vector of gene i; and $S_{j,t} \in \mathbb{R}$ is the jth element of the cell state at time t.

Gene Activation. The gene activation function of IMRO regulatory and output genes is similar to the FGRN's activation function, but removes the need for arbitrary constants. The activation function of gene i is defined by its scale α_i and its threshold $\theta_i \in [-1, 1]$. For gene i at time t, the activation $a_{i,t} \in [0, 1]$ is given by

$$a_{i,t} = \begin{cases} \frac{\max(v_{i,t}, \theta_i) - \theta_i}{1 - \theta_i} & \text{if } \theta_i > 0 \\ \frac{\min(v_{i,t}, |\theta_i|)}{|\theta_i|} & \text{if } \theta_i < 0 \end{cases}, \text{where } v_{i,t} = \frac{\tanh(\alpha_i m_{i,t}) + 1}{2}$$

This allows for a large variety in the direction and scale of the activation function, while preserving the general shape of its natural equivalent : a 0 or 1 plateau, followed or preceded by a smooth curve to/from the other end of the $[0, 1]$ range. This function also preserves both the digital aspect of natural gene activation (a gene can be activated or not), and the analog aspect (once activated, a variable amount of protein can be produced, depending on the activation level).

Protein Output. When in a regulatory gene, the protein output component generates a protein on activation (when the activation value is non null). The component defines the protein's L level array, as well as its initial time-to-live τ. The protein's value v is determined from the combination of a scaling factor β and the activation value. For gene i at time t, the protein value is $v_{i,t} = \beta_i a_{i,t}$, and similarly when in an input gene, except an external scalar input then replaces the activation value. In randomly initialised genomes, the protein output components of input genes are initialised with a τ of one, and only have one positive value amongst the levels, to avoid a flooding of the cell state.

Scalar Output. The scalar output component determines the return value of an output gene. If a boolean value is desired, the output of gene i at time t is $o_{i,t} = 0$ iff $a_{i,t} = 0$, otherwise 1. If a real value is desired, the component has a scale parameter β, and a threshold parameter T; the output of gene i at time t is then $o_{i,t} = \max(a_{i,t}, 0)$ iif $T \geq 0$, and $|T| - \min(a_{i,t}, |T|)$ otherwise.

5 Experiments

In this section we first give some relevant parameter settings and experimental details, before presenting some preliminary experiments along with their results, and finally detailing the results of the pole balancing and acrobot experiments.

5.1 Parameter Settings and Experimental Details

A maximum of 10,000 genomes are evaluated per run for the preliminary experiments and the pole balancing. For the acrobot, following ref [4], a maximum of 3,000 genomes is evaluated per run. All experiments are run 50 times.[1]

Genetic Algorithm. IMRO and FGRN genomes are evolved using the ALPS genetic algorithm[7] with a layer size of 25 and an age gap of 10. Tournament selection is used in each layer, with a tournament size of 4 and with elitism set to 3. Parents are selected from the top 40% of each layer, except in one percent of cases, where a parent is selected randomly. The gene component mutation rate is 0.1, and uniform crossover is always applied. ALPS was found to increase the reliability with which successful FGRN controllers were found[10].

[1] The source code for all the experiments and systems described in this paper is available at `http://github.com/susano/ppsn2012`

FGRN. The FGRN genomes evolved are composed of four regulatory genes, one receptor gene, one behavioural (output) gene, and as many environmental genes as there are inputs. The zero centered input-mapping[10] is used.

IMRO. The IMRO genomes evolved are composed of two regulatory genes, one output gene, and as many input genes as required by the problem. The size of the cell state N is set to eight. The allocation of more regulatory genes to FGRN genomes than to IMRO genomes followed preliminary experiments in which FGRN performances were poorer with fewer than four regulatory genes.

5.2 Preliminary Experiments

The initial test[2] of the FGRN model's developmental capabilities was to attempt to evolve genomes able to produce specific activation patterns (see Figure 3), no input was given. The fitness of a genome was the number of matches between its activation output and the pattern. The ability to generate a variety of activation patterns, independently of any input, can allow the exploration of otherwise closed regions of the space of possible controllers, and we therefore applied both FGRN and IMRO genomes to this task.

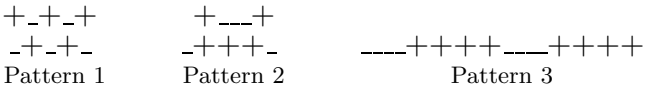


Fig. 3. Test activation patterns from ref [2]. Patterns 1 and 2 require two separate output genes per genome

Table 1. The percentage of successfully generated patterns, and the mean number of evaluations required to success (standard deviation in parenthesis).

	FGRN		IMRO	
Pattern 1	100%	810(894)	100%	275(244)
Pattern 2	100%	619(562)	100%	398(364)
Pattern 3	34%	6095(2797)	100%	1794(1758)

The results are impressive: IMRO genomes can be evolved significantly faster ($p < 0.001$) to produce the desired pattern than FGRN genomes, and in the case of pattern 3, much more reliably. (It should be noted that the FGRN results on pattern 3, despite being significantly worse than the IMRO results, are an improvement on Bentley's initial results for this pattern [2], where an additional guidance component needed to be added to the fitness to successfully evolve this pattern. We attribute this to the use here of the ALPS genetic algorithm, and to an improvement in the FGRN settings we use[10])

5.3 Pole Balancing and Acrobot Experiments

The setup of the pole balancing and acrobot problems is detailed in Section 3. The controllers are run on the pole balancing problems for 100,000 simulated timesteps (≈ 30 minutes). For the acrobot, each controller is run for a maximum of 4,000 timesteps; this was necessary instead of the 1,000 timesteps to allow the genetic algorithm to find initial solutions from which to start improving. The fitness for pole balancing is the number of timesteps before a pole falls down. For the acrobot it is the number of timesteps until swing-up, inversed.

Pole balancing. The results are detailed in Table 2. Both FGRN and IMRO genomes were able to evolve successful controllers at every run for the single pole balancing problem, but only IMRO genomes were able to evolve the ability to solve the double-pole balancing problem, the most successful FGRN controller only balancing the poles for 172 timesteps (≈ 3 seconds) out of 100,000.

Table 2. Number of failures/evaluations before a successful controller is found. Key: SD = Standard Deviation

	FGRN			IMRO		
	Mean(SD)	Best	Worst	Mean(SD)	Best	Worst
Single Pole	729(787)	50	5129	480(356)	33	1985
Double Pole	-	-	-	2200(1486)	487	8155

Acrobot. Table 3 details the results of the IMRO and FGRN systems on the acrobot, as well as those of the SARSA-RGD system, an online learning method, and of LSPI, a policy iteration method, on the same problem. The IMRO system performed significantly better than both the FGRN system and LSPI ($p < 0.001$), finding on average significantly shorter trajectories. But it performed worse than SARSA-RGD, though SARSA-RGD was less reliable, failing in some of the runs to find any swing-up trajectory. Figure 4 shows the trajectory of an acrobot controlled by the IMRO system.

Table 3. Length of the shortest trajectory found to acrobot swing-up, sorted by shortest average trajectory. The results for SARSA-RGD and LSPI are taken from ref [4].

	Mean(SD)	Best	Worst
SARSA-RGD	276.56(106.62)	238	-
IMRO	307.68(40.42)	266	500
LSPI	335.90(12.11)	315	343
FGRN	357.26(69.92)	257	588

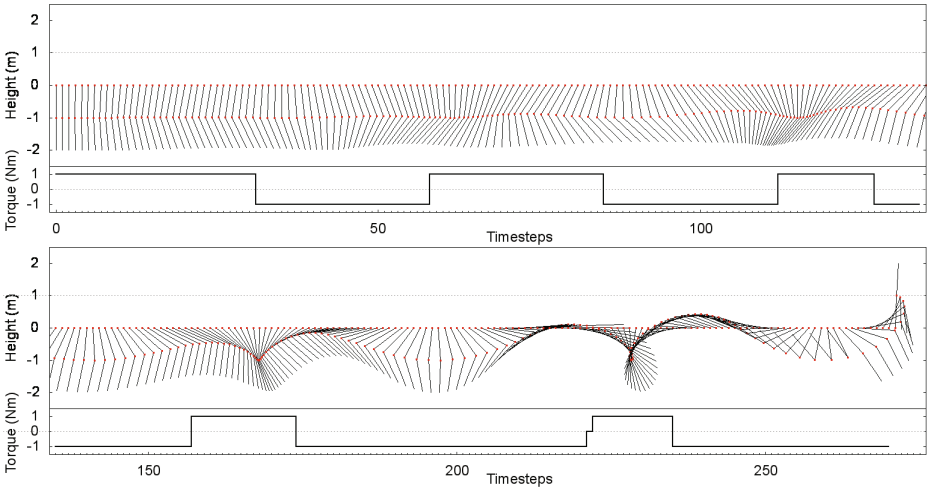


Fig. 4. An example acrobot swing-up trajectory produced by the IMRO system. Top, the position of the acrobot at each time step. Bottom, the force applied at each timestep. Long periods of the same activation, and limited use of the null force action, are typical of efficient swing-up solutions.

6 Discussion

This paper has introduced IMRO, a simplified abstraction of the GRN model underpinning the successful but complex FGRN system. These simplifications, already desirable in themselves, resulted in greatly improved performance on control tasks of a widely different nature: while the pole balancing is a stabilisation problem, the acrobot is the exact opposite, requiring the controller to destabilise the system until it reaches a remote region of the state-space.

The performance of the FGRN system in the same experiments, and particularly the combined failure on the double pole balancing problem and in the generation of pattern 3, and its limited success on the acrobot, lead us to believe the FGRN system to be more suitable for control problems not requiring very precise control sequences, but having a large variety of possible complex control strategies. This might find its root in the original developmental nature of the FGRN system, where the complexity of the system might be more useful.

Future work with the IMRO system will focus initially on finding harder control problems to which it can be applied. If it proves necessary to make changes to the system this will be facilitated by its modularity and the clearly defined interfaces between its components. However we do not seek complexity for its own sake, but to abstract from biology only those elements that prove useful in problem solving, which may indirectly also throw light on why nature's solutions to problems are frequently so effective.

References

1. Bentley, P.J.: Evolving Fractal Gene Regulatory Networks for Robot Control. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) ECAL 2003. LNCS (LNAI), vol. 2801, pp. 753–762. Springer, Heidelberg (2003)
2. Bentley, P.J.: Fractal Proteins. Genetic Programming and Evolvable Machines Journal 5, 71–101 (2004)
3. Boone, G.: Efficient reinforcement learning: Model-based acrobot control. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 229–234. IEEE (1997)
4. da Motta Salles Barreto, A., Anderson, C.W.: Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. Artificial Intelligence 172(4-5), 454–482 (2008)
5. Funkquist, J., Stephan, V., Schaffernicht, E., Rosner, C., Berg, M.: SOFCOM-Self-optimising strategy for control of the combustion process. VGB PowerTech, 49 (2011)
6. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated Neural Evolution through Cooperatively Coevolved Synapses. The Journal of Machine Learning Research 9, 937–965 (2008)
7. Hornby, G.S.: ALPS: The Age-Layered Population Structure for Reducing the Problem of Premature Convergence. In: GECCO 2006, pp. 815–822. ACM, New York (2006)
8. Joachimczak, M., Wróbel, B.: Evolving gene regulatory networks for real time control of foraging behaviours. In: Artificial Life XII, pp. 348–355 (2010)
9. Krohn, J., Bentley, P.J., Shayani, H.: The Challenge of Irrationality: Fractal Protein Recipes for PI. In: GECCO 2009, Montreal, Canada (July 2009)
10. Krohn, J., Gorse, D.: Fractal Gene Regulatory Networks for Control of Nonlinear Systems. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6239, pp. 209–218. Springer, Heidelberg (2010)
11. Lopes, R.L., Costa, E.: ReNCoDe: A Regulatory Network Computational Device. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) EuroGP 2011. LNCS, vol. 6621, pp. 142–153. Springer, Heidelberg (2011)
12. Nicolau, M., Schoenauer, M., Banzhaf, W.: Evolving Genes to Balance a Pole. In: Esparcia-Alcázar, A.I., Ekárt, A., Silva, S., Dignum, S., Uyar, A.Ş. (eds.) EuroGP 2010. LNCS, vol. 6021, pp. 196–207. Springer, Heidelberg (2010)
13. Spong, M.W.: The swing up control problem for the acrobot. IEEE Control Systems Magazine 15(1), 49–55 (1995)
14. Sutton, R.S.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Advances in Neural Information Processing Systems, pp. 1038–1044 (1996)
15. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction, vol. 28. Cambridge Univ. Press (1998)
16. Zahadat, P., Christensen, D.J., Schultz, U.P., Katebi, S., Stoy, K.: Fractal Gene Regulatory Networks for Robust Locomotion Control of Modular Robots. In: Doncieux, S., Girard, B., Guillot, A., Hallam, J., Meyer, J.-A., Mouret, J.-B. (eds.) SAB 2010. LNCS, vol. 6226, pp. 544–554. Springer, Heidelberg (2010)
17. Zahadat, P., Katebi, S.D.: Tartarus And Fractal Gene Regulatory Networks With Inputs. Advances in Complex Systems (ACS) 11(06), 803–829 (2008)