A Gene Regulatory Network Model for Control

Jean-Baptiste Krohn

University College London

Dept. of Computer Science

Gower Street

London WC1E 6BT
United Kingdom

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London (UCL).

Department of Computer Science

University College London

Abstract

The activity of a biological cell is regulated by interactions between genes and proteins. In artificial intelligence, this has led to the creation of developmental gene regulatory network (GRN) models which aim to exploit these mechanisms to algorithmically build complex designs. The emerging field of GRNs for control aims to instead exploit these natural mechanisms and this ability to encode a large variety of behaviours within a single evolvable genetic program for the solution of control problems.

This work aims to extend the application domain of GRN models to previously unsolved control problems; the focus will here be on reinforcement learning problems, in which the dynamics of the system controlled are kept from the controller and only sparse feedback is given to it. This category of problems closely matches the challenges faced by natural evolution in generating biological GRNs. Starting with an existing GRN model, the fractal GRN (FGRN) model, a successful application to a standard control problem will be presented, followed by multiple improvements to the FGRN model and its associated genetic algorithm, resulting in better performances in terms of both reliability and speed. Limitations will be identified in the FGRN model, leading to the introduction of the Input-Merge-Regulate-Output (IMRO) architecture for GRN models, an implementation of which will show both quantitative and qualitative improvements over the FGRN model, solving harder control problems. The resulting model also displays useful features which should facilitate further extension and real-world use of the system.

I, Jean-Baptiste Krohn confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

To my father and mother

Acknowledgements

Many have helped in the realisation of this work, but I am particularly grateful to the following people for their contribution to the completion of this thesis.

Denise Gorse supervised my work and through our frequent discussions managed the impressive feat of directing my thoughts and energies into properly structured academic research, which she then taught me to put down into readable prose. Chris Clack's concise and accurate comments and feedback throughout the PhD were most enlightening and proved invaluable towards finalising it.

I am grateful to Peter Bentley for his supervision of my initial work and introducing me to fractal gene regulatory networks. I would also like to thank Mark Herbster for his patience and advice. My viva examiners, Miguel Nicolau and David Clark, provided through their challenging, well thought-out, questions and researched criticism a memorable, and ultimately deeply satisfying, experience. I enjoyed comradeship in research at UCL with Ghada Hassan and Domenico Cozzetto.

I am grateful to Lee for his friendship, support, and sensible advice throughout this whole adventure. Among the friends who have encouraged me during this process, I must also single out Annora Eyt-Dessus and Dan Bratton for their continued and unwavering support, Henry Potts for his wise advise, and particularly Maria-Angelica Bălă for her support and the light and gaiety she brings to tough times. Foremost I thank my mother and father for always encouraging me to learn and better myself.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Biology has often inspired the design of machine learning systems; from artificial neural networks to evolutionary computing, biological metaphors have provided a useful basis. Similarly, this work looks at gene regulatory networks (GRNs) as inspiration for control. GRNs are composed of the set of interactions between genes and proteins within a biological cell, and act as controllers in situations as different as single cell bacteria and multiple-cell organisms, with orders of magnitude of variation in complexity and 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).

This versatility of natural GRNs lead to the initial introduction of artificial GRN (AGRN) models for the purpose of solving a different problem, that of development. In an AGRN-based developmental system, genomes are evolved via a genetic algorithm (GA). The fitness of a genome is determined indirectly, the genome being iteratively 'executed' as a GRN, and the output of the GRN being the final product on which the fitness is evaluated. However, though these developmental uses are valid applications of the GRN metaphor, at the cellular level natural GRNs act as the cell's controller. Comparatively little work has been based on using an AGRN for direct control. Direct control means here feeding the control inputs in the AGRN and using the AGRN's outputs as control actions, as opposed to indirect control which consists in using the AGRN to developmentally generate a controller (e.g. a neural network). For direct control, AGRN genomes are evolved to maximise some measure of success in controlling a given system.

Work using AGRN models as direct controllers includes the BioSys AGRN model from Quick et al., which was applied to two basic control problems: dampening the variations of temperature in a thermostat-like system, and generating light-following behaviour in a wheeled robot [QNDR03]. More recently, Nicolau et al. modified Banzhaf's artificial regulatory network (ARN) model to apply it to the classic pole balancing problem [NSB10]; and Joachimczak and Wróbel applied a GRN model they initially introduced for developing three-dimensional morphologies to the control of foraging agents in a virtual world [JW10].

Fractal GRN (FGRN), an AGRN model revolving around the idea of using square portions of the Mandelbrot fractal as proteins to mimic the complexity of natural protein interactions, was introduced by Bentley [Ben04b]. The model was first applied to simple developmental tasks, such as binary pattern

generation [Ben04b], function approximation [Ben05]; and to the control task of generating collision-avoiding, wall following behaviours in a toy robot[Ben03a]. Zahadat et al. applied the FGRN model to a grid-world, box-pushing, control problem [ZK08], and to the distributed locomotion control of modular robots [ZCS+10]. The variety of control problems the FGRN model has been applied to makes it the most promising candidate for improving direct AGRN-based control.

This work focuses on reinforcement learning (RL) control problems, in which the controlled system's dynamics are completely unknown and the reinforcement feedback is limited. Keeping the dynamics unknown to the controller is important for real world applications (e.g. coal furnace combustion control [FSS+11]), and minimising domain knowledge helps ensures the wide applicability of the method.

Evaluating a controller's performance on a RL problem consists of running a loop in which the controller is given inputs from the controlled system (e.g. the state of the system), and must return an output vector that specifies actions to be taken; the controller may then receive a scalar reinforcement signal from the controlled system (a higher value corresponding to a higher reward). The evaluation ends when a failure condition is reached, or when a given time period has expired. A successful controller must maximise the sum of reinforcements over the course of its evaluation. When evolving AGRN controllers for RL, the sum of reinforcements received by a controller in an evaluation is the fitness of the genome which produced the controller. The AGRN controller itself does not receive the reinforcement signal.

1.1 Research problem

It is the hypothesis of this work that improvements in the FGRN model will allow the solving of RL problems currently unsolvable by the FGRN, and unsolved by any other AGRN-based method. And that these improvements will also provide an increase in performance by decreasing the number of controller evaluations required by the FGRN model to find a successful solution on a range of RL problems.

This work is believed to be significant, as it will advance the field of AGRN-based control, partly closing the gap of applicability between AGRNs and traditional control techniques. The interest in AGRN has been increasing in the last decade; more specifically the number of publications on direct control using AGRN has increased in recent years, mostly as pre-existing GRN models for development are modified to be applied to control problems. An example of this interest is the creation at the Eleventh European Conference on Artificial Life (ECAL 2011) of a Workshop on the Design, Simulation, Construction and Testing of Synthetic Gene Regulatory Networks for Computation, Control and Communication (SynBioCCC).

This work is believed to be difficult. The FGRN model is complex, and contains poorly documented arbitrary constants. Randomly generated fractal proteins tend to be of limited use and for better results Bentley recommends initialising the proteins in the genome by drawing from a pre-evolved, limited set of proteins [Ben04b]. These issues, added to the opacity of the resulting controllers, are likely to

limit the applicability of the model, and make it harder to understand and modify in a targeted manner. Compared to some other AGRN models, the FGRN model is also computationally inefficient, making experimentation slower.

This work is believed to be achievable. As detailed above, the FGRN has a number of issues and fixing these issues is likely to lead not only to quantitative improvements in reducing the number of controller evaluations required to find a successful solutions on a range of problems, but also to qualitative improvements allowing the system to solve problems which previously could not be solved with it.

1.2 Contributions

The contributions in this work will be:

- Advanced demonstrations of the evolvability of the FGRN model on a developmental problem, as a basis for the subsequent work on control problems.
- First application of the FGRN model to a standard RL control problem: pole-balancing.
- Introduction of improved, simplified, non-fractal chemistries for the FGRN model, leading to a decrease in the number of controller evaluations required to reach a successful solution on the pole-balancing problem.
- Further simplifications of the FGRN model, leading to the introduction of a fast, modular GRN model able to solve control problems previously unsolvable with the FGRN model, and unsolved by any other evolutionary method.

1.3 Thesis outline

This thesis is structured as follows. Chapter 2 provides background on the control problems studied, natural GRNs, and artificial GRN models. In Chapter 3 the FGRN model for development is described in detail, and results on a developmental problem are presented, illustrating the ability of the FGRN model combined with a genetic algorithm to refine genomes to produce target activations pattern with arbitrary criteria. Chapter 4 describes the alterations to the FGRN model required to its application to control problems, before showing the results of its application to multiple versions of the pole balancing problem; improvements to the speed and reliability with which a successful controller is found are then presented. The subject of Chapter 5 is an investigation into protein encoding in the FGRN model: the important characteristics and limitations of the fractal process of generating proteins (the fractal protein encoding) are identified, and alternative encodings mitigating these limitations are presented and found to improve performance on the pole balancing control problem. In Chapter 6, based on the findings of Chapter 5, a simpler, faster, modular GRN architecture and model are presented and found to successfully generate controllers for the double pole balancing problem, which was not solvable with the FGRN model, and a hardened version of the acrobot problem, for which no evolutionary solution existed previously. In Chapter 7, the IMRO system introduced in Chapter 6 is combined with the best

performing protein encoding designed in Chapter 5, and the IMRO system's applicability is validated on variations of problems studied in previous chapters, and on an additional control problem. Chapter 8 concludes this thesis, summarising its findings and lists several avenues of future work.

1.4 Publications

This thesis incorporates material from the following publications:

- J. Krohn, P.J. Bentley, and H. Shayani. The Challenge of Irrationality: Fractal Protein Recipes for PI. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009)* pages 715–722, ACM, 2009.

- J. Krohn and D. Gorse. Fractal Gene Regulatory Networks for Control of Nonlinear Systems. In *Parallel Problem Solving From Nature (PPSN XI)*, volume 6239 of *Lecture Notes in Computer Science (LNCS)*, pages 209–218, Springer, 2010.

- J. Krohn and D. Gorse. Extracting Key Gene Regulatory Dynamics for the Direct Control of Mechanical Systems. In *Parallel Problem Solving From Nature (PPSN XII)*, volume 7491 of *Lecture Notes in Computer Science (LNCS)*, pages 468–477, Springer, 2010.

Chapter 2

Literature Review

In this chapter, the application domain, reinforcement learning control is first described. The basics of natural gene regulatory networks (GRNs) are then covered, followed by a review of the existing evolutionary GRN models which they inspired, with a focus on GRN models for control.

2.1 Reinforcement learning control

Reinforcement Learning (RL) control consists of discovering what actions to take for any given environmental state in order to maximise a scalar reward [SB98]. Solving reinforcement learning problems generally requires a balance between exploration (attempting new actions leading to yet unknown consequences) and exploitation of existing knowledge to maximise reinforcement. Examples of reinforcement problems, the pole balancing, acrobot, and mountain car tasks are detailed in the following sections of this chapter. The difficulty of reinforcement learning stems from the limited information given to the controller about the system controlled, and the limited learning feedback received by the controller. The dynamics of the controlled system are kept completely hidden, the only information on the problem given to the controller being often limited to the number and range of inputs and outputs. As opposed to supervised learning approaches, no direct feedback is given to the controller as to any of its actions being good or bad. The feedback given, or reinforcement, is a function of the situation of the system controlled, and not the controller's latest action, and can be very sparse (as is the case for the problems studied in this work). The constraints not only make the problems more difficult, but also ensure a wide applicability of the learning controllers, which cannot rely on specific knowledge of the problem. Figure 2.1 illustrates these interactions between the controller and the controlled system.

2.1.1 Genetic reinforcement learning

As opposed to traditional RL approaches, which explicitly learn a value surface (a mapping attributing a desirability value for each combination of a system state $\mathbf{x}$ and output $\mathbf{y}$), genetic reinforcement learning (a term introduced by Whitley et al. [WDDA93]) uses metaheuristics, typically and in this work genetic algorithms (GAs), to produce candidate controllers which are each tested in turn. The fitness of a candidate controller is then the sum of reinforcements over the time of its evaluation. The exploration role is fulfilled by the initial generation of random controllers, and the subsequent variation (mutation and crossover) in the population, while the selective nature of GAs ensure the prevalence of controllers

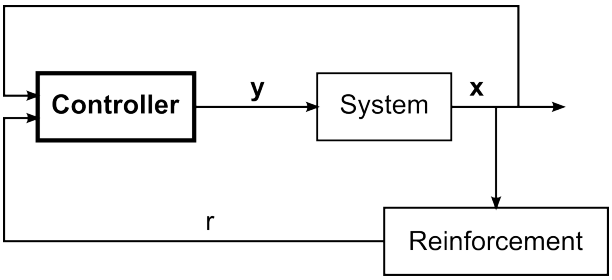


Figure 2.1: The interactions between controller and controlled system. The controller is given as input the state vector **x** of the system and outputs an action vector **y**; it also receives at each control iteration a reinforcement scalar **r** which is function of **x**.

which maximise the sum of reinforcements.

2.1.2 The pole balancing problem

Pole balancing is a well-known and well-studied control problem that has been used as a benchmark for the design and test of many controllers [Ige03]. It has previously been used as a means to evaluate and develop control systems before applying them to real-world control tasks (e.g. Gomez and Miikkulainen's neuroevolution system for fin-less rocket guidance [GM03]). The system controlled is composed of a free-swinging pole attached on top of a cart, which can itself move left or right on a track (see Figure 2.2). The aim is to stop the pole from falling down only by giving the cart a small push left or right at each timestep, while keeping the cart within the boundaries of the track, for half an hour. The input of the controller at each time step is the state of the system, and the output is a boolean value determining whether the cart is pushed left or right by a fixed force ('bang-bang' control). The state of the system $< x, \theta, \dot{x}, \dot{\theta} >$ is composed of the angle of the pole to the vertical 'θ', of the distance of the centre of the cart to the centre of the track 'x', and of their reciprocal velocities. The reinforcement provided to the controller is $+1$ for each timestep until failure, which happens if the angle of the pole to the vertical is too high, or the cart goes outside the boundaries of the track. Multiple harder variants of the problem exist; the most popular among them is the double pole balancing problem which adds a second smaller pole to be balanced on the cart.

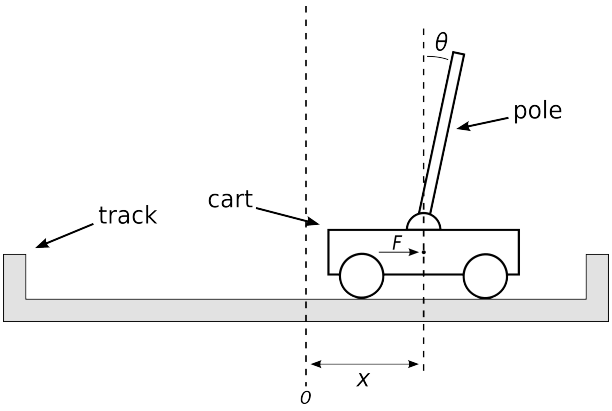
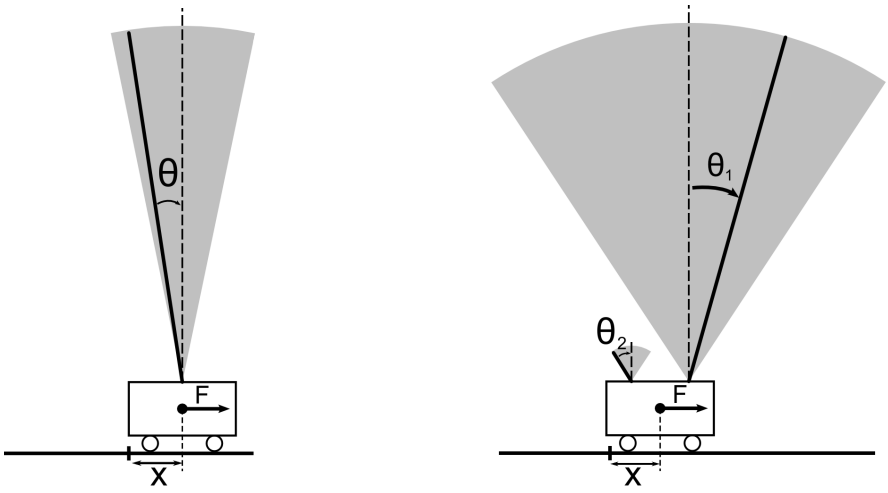


Figure 2.2: Pole balancing: the cart-pole-track system



(a) Single pole balancing

(b) Double pole balancing

Figure 2.3: The single and double pole balancing problems. While the poles are balanced within the greyed out areas, a fixed positive reinforcement is given. The pole sizes and accepted angle ranges are to scale.

Formal problem description

The variations of the pole balancing considered in this work will be the traditional version, here titled single pole balancing (SPB); and the commonly used double pole balancing (DPB). Both problems are illustrated in Figure 2.3. Single pole balancing is usually (and here) defined to be the problem of keeping the angular position θ of the 1.0m hinged pole within 12° of vertical, and the distance h 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 is applied to the cart at each time step). There are fifty control timesteps per second, and the fitness of a controller is the number of timesteps for which the pole was balanced before failure. The controller is considered to be successful after balancing the pole for at least a hundred thousand timesteps (equivalent to approximately thirty minutes). The double pole balancing problem is similar, differing in the addition of an independently swinging 0.1m pole on the cart, that must also be balanced. It is generally thought to be considerably more difficult. To not allow the poles to move in concert, thereby simplifying the problem of controlling them, the longer pole's starting position is at an angle of 4°, with null velocity. The maximum acceptable angle of the pole from the vertical is brought from 12° to 36°, to make the problem solvable.

The equations of motion and constant values used here are adapted from Gomez et al [GSM08], these represent the most commonly used version of the problem and describe both the single and double pole systems. The equations of motion for N poles balanced on a single cart are

$$\ddot{h} = \frac{F - \mu_c \mathrm{sgn}(\dot{h}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{h}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right)$$

Table 2.1: Variables associated with the pole balancing equations of motion

Symbol	Description	Range
F	Force applied to the cart	$-10, +10$N
x	Position of the cart on the track	$[-2.4, 2.4]$m
θ_1	Angle of the first pole from vertical	SPB: $[-12, 12]^\circ$ DPB: $[-36, 36]^\circ$
θ_2	Angle of the second pole from vertical	DPB: $[-36, 36]^\circ$

Table 2.2: Constants associated with the pole balancing equations of motion

Symbol	Description	Value
g	Gravity	-9.8m.s^{-2}
M	Mass of the cart	1.0kg
m_i	Mass of the i^{th} pole	m_1 = 0.1kg m_2 = 0.01kg
l_i	Half length of the i^{th} pole	l_1 = 0.5m l_2 = 0.05m
μ_c	Coefficient of friction of the cart on the track	0.00005
μ_{pi}	Coefficient of friction of the i^{th} pole hinge	0.000002

where $\tilde{F}_i$ is the effective force from the i^th pole,

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g \sin\theta_i \right)$$

and $\tilde{m}_i$ is the effective mass of the i^{th} pole,

$$\tilde{m}_i = m_i \left(1 - \frac{3}{4} \cos^2\theta_i \right)$$

The variables of the system and the range of value they can take are detailed in Table 2.1. The constants M, m_i, l_i, μ_c, μ_{pi} are defined in Table 2.2. As in ref [GSM08], two-step fourth order Runge-Kutta, a common method for the approximation of ordinary differential equations, is used with a 0.02s timestep to integrate the equations of motion.

The state given as input for to the controller at each timestep is $< x, \theta, \dot{x}, \dot{\theta} >$ for the single pole balancing problem, and $< x, \theta_1, \theta_2, \dot{x}, \dot{\theta_1}, \dot{\theta_2} >$ for double pole balancing. Pole balancing problems have also often be made harder by depriving the controller of any velocity information; controller inputs are then $< x, \theta >$ (SPB), and $< x, \theta_1, \theta_2 >$.

Previous solutions

The original single pole problem was first solved with a value surface-based method by Barto et al. [BSA83], with Wieland the first to evolve neural networks for the control of the system [Wie90]

(for the single, jointed, and double pole versions of the problem), but with additional fitness feedback which penalised any departure from the balanced position.

Whitley et al. presented the first evolutionary solution to the single pole balancing problem with the fitness function used in this work (the time from start until system failure) [WDDA93]; for this the weights of neural network controllers were evolved with a GA. Whitley et al. were also the first to investigate, in that work, the ability to generalise of the final successful controllers, by testing them against a vast range of initial cart and pole positions. Aiming to maximise this ability of the generated controllers to control the system from a large range of starting positions, each control run during the evolutionary process was started from a different, randomly generated, starting position.

Subsequent solutions were achieved using a variety of other methods such as genetic programming [SF99]. More recently, neuroevolution methods of increasing sophistication have used various versions of the pole balancing problem for validation and as a benchmark [Ige03]; in 2008, Gomez et al. produced an extensive comparison of the performance of current methods on the single and double pole balancing problems [GSM08]. In terms of the speed with which a successful controller is found, neuroevolution, a genetic reinforcement learning method evolving the weights and structure of neural network controllers, is currently the most successful approach. Gomez et al.'s table of results for the single, full-state, pole balancing is reproduced in Table 2.3; it should be noted that there were some variation in the problem settings, some methods being allowed to apply a variable force on the cart instead of the fixed amount in bang-bang control. Note that all experimental settings linked to the single and double pole balancing have been taken from that work [GSM08], to allow for ease of comparison with their results, but using only the harder bang-bang control with no possibility of variable force output.

2.1.3 The acrobot swing-up problem

The acrobot is a two-link underactuated robot; it is roughly analogous to a gymnast hanging by the hands from a fixed bar with arms and legs straight, and only able to act by bending at the hips; it is illustrated in Figure 2.4. The aim of the gymnast is to swing her body above the horizontal bar [Spo95], in preparation to a handstand. The acrobot system is much simplified, with the solid links freely rotating around both the bar and the hip joint. The controller must then, taking as input the full state of the system, decide at each timestep what torque to apply to the hips joint.

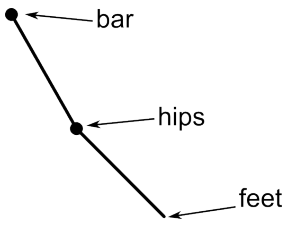


Figure 2.4: The inspiration for the acrobot problem. A gymnast hanging from an horizontal bar, attempting to swing her body above the bar through only hips actions.

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

Table 2.3: 2008 table of average number of evaluations needed to obtain a controller able to solve the full state single pole balancing problem for 100,000 timesteps, over 50 runs. Note that apart from the CMA-ES [Ige03] and the AHC solutions which use bang-bang control, all other solutions listed in this table allow controllers to specify a variable, continuous, amount of force to the cart. Reproduced from Gomez et al. [GSM08], with the addition of the family to which each of these methods belongs.

Method	**Evaluations**	**Family**
CoSyNE	98	Neuroevolution
ESP	289	Neuroevolution
NEAT	743	Neuroevolution
SANE	302	Neuroevolution
CNE	352	Neuroevolution
CMA-ES	283	Evolutionary strategy
SARSA-CABA	965	Q-Learning
SARSA-CMAC	540	Q-Learning
Q-MLP	2,056	Q-Learning
PGRL	28,779	Reinforcement learning
AHC	189,500	Reinforcement learning

next. However the problem goals can be put into two broad categories:

- **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, 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. Solutions to the swing-up problem have frequently also involved pre-existing domain knowledge. However Sutton successfully applied a combination of SARSA with coarse input coding the swing up problem [Sut96]. The control actions followed a bang-zero-bang scheme: the torque applied to the middle joint was either 1Nm, -1Nm, or no torque. The frequency of control was 5Hz (the controller was polled for an action five times per second). More recently, da Motta Salles Barreto and Anderson [dMSBA08] introduced a harder version of the acrobot swing-up problem, based on Sutton's, by multiplying by four the frequency of control actions (using a 20Hz rather than 5Hz control frequency). This makes the problem harder, requiring the controller to generate a much longer series of consistent actions.

This work will focus on this harder version of the acrobot swing-up problem, described in detail below. The description is followed by a review of existing solutions of earlier versions of the acrobot

problem. These solutions, as opposed to the approach in this work, generally incorporate extensive knowledge of the problem (e.g. dynamics, desirable level of the system's energy, including the height reached by a controller within the controller's fitness function, etc.).

Formal problem description

Figure 2.5 illustrates the acrobot swing-up problem and the angular values which form the state of the system.

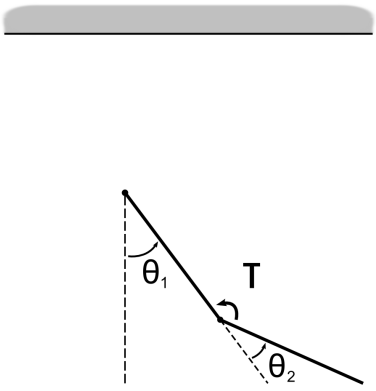


Figure 2.5: The acrobot swing-up problem. The aim is for the controller to guide the tip (feet) of the acrobot more than one link-height above the horizontal bar in the least amount of time. The tip of the acrobot much reach the greyed area on top for a swing-up to be considered successful.

The equations of motion used here are the same as those used by Sutton [Sut96]. The system's state is entirely defined by the two angles θ_1 and θ_2 and their associated velocities $\dot{\theta_1}$ and $\dot{\theta_2}$. The action chosen by the controller is a torque $\tau \in \{+1, -1, 0\}$ which is applied to the joint between the two links. This action τ, and the current state of the system $< \theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2} >$ determine the angular accelerations $\ddot{\theta_1}$ and $\ddot{\theta_2}$ according to the following equations:

$$\ddot{\theta_1} = -d_1^{-1}(d_2\ddot{\theta_2} + \phi_1)$$

$$\ddot{\theta_2} = \left(m_2 l_{c2}^2 + I_2 - \frac{d_2^2}{d_1}\right)^{-1} \left(\tau + \frac{d_2}{d_1}\phi_1 - \phi_2\right)$$

where

$$d_1 = m_1 l_{c1}^2 + m_2(l_1^2 + l_{c2}^2 + 2 l_1 l_{c2} \cos\theta_2) + I_1 + I_2$$

$$d_2 = m_2(l_{c2}^2 + l_1 l_{c2} \cos\theta_2) + I_2$$

$$\phi_1 = -m_2 l_1 l_{c2} \dot{\theta_2}^2 \sin\theta_2 - 2 m_2 l_1 l_{c2} \dot{\theta_2}\dot{\theta_1} \sin\theta_2 + (m_1 l_{c1} + m_2 l_1) g \cos(\theta_1 - \pi/2) + \phi_2$$

$$\phi_2 = m_2 l_{c2} g \cos(\theta_1 + \theta_2 - \pi/2)$$

Table 2.4: Descriptions and values of the constant parameters in the acrobot problem

Symbol	Description	Value
l_1, l_2	Link lengths	1.0m
l_{c1}, l_{c2}	Link lengths to centre of mass	0.5m
I_1, I_2	Link moments of inertia	1.0kg.m^2
g	Gravity	9.8m.s^{-2}

The role and value of the constants in the equations above are detailed in Table 2.4. As in Sutton's work, the angular velocities are bound as follows : $\theta_1 \in [-4\pi, +4\pi]$, $\theta_2 \in [-9\pi, +9\pi]$. As in da Motta Salles Barreto and Anderson's work [dMSBA08], the control frequency used is 20Hz, making the problem harder. The two-step Runge-Kutta fourth order method is used to integrate the equations of motion. The starting position of the system is equivalent to that of the acrobat hanging from the bar at rest, with both angles θ_1 and θ_2 set to zero.

Before being sent as input to the controller, the state of the system $< \theta_1, \theta_2, \dot{\theta_1}, \dot{\theta_2} >$ is pre-processed. As the joints are allowed to fully rotate, the issue of the mapping of θ_1 and θ_2 onto a finite range appears; should it be $[0, 2\pi]$, $[-\pi, +\pi]$, some other variation? To provide fully continuous input states, even in the case of a complete rotation and avoid any human bias in the choice of the range, the $\sin$ and $\cos$ of each angle are given as input in replacement of θ_1 and θ_2. Note that this transformation is not specific to the acrobot, but to angular inputs which can loop cover the full range of possible angles. With the angular velocities, the full input given to the controller at each timestep is therefore: $\sin\theta_1$, $\cos\theta_1$, $\sin\theta_2$, $\cos\theta_2$, $\dot{\theta_1}$, and $\dot{\theta_2}$.

As in Sutton's and da Motta Salles Barreto and Anderson's work the only reinforcement given during the evaluation of a controller is -1 for each timestep at which the goal is not achieved, and the evaluation is stopped when either the goal is reached, or a maximum number of timesteps has lapsed.

Previous solutions

Contrary to the pole balancing problem and as opposed to the previously mentioned works of Sutton [Sut96] and da Motta Salles Barreto and Anderson [dMSBA08], most methods employed to find successful acrobot control strategies have required extensive knowledge of the dynamics of the acrobot system, and detailed feedback of the proximity of the goal state. This is also opposed to the 'black box' approach taken in this work to not use any knowledge of the controlled system within the initial setup of the controller beyond the number of inputs and outputs; and while the controller is running, to provide only minimal feedback on failure/success. Initial solutions to the acrobot swing up problem were control strategies derived from the problem's dynamics [Spo95] [BP97]. From the acrobot dynamics, Brown and Passino also develop Fuzzy controllers [BP97]: Genetic algorithms are used, via a complex fitness function, to tune the parameters of fuzzy controllers.

Lookahead search solutions

Boone devised a lookahead search algorithm, for swing up which targeted the level of energy of the system in the target position [Boo97b]; it used full knowledge of the system's dynamics to estimate the energy level of future possible states. Like in Sutton's work [Sut96], the control frequency was 5Hz; however Boone used bang-bang instead of bang-zero-bang control, likely to reduce the width of the search tree to be considered at each timestep. Once the desired level of energy is reached, the lookahead window size is doubled and the target of the search algorithm changes to finding a state roughly equal to the handstand position. Boone's work was successful and is interesting as it can likely be applied to the control of a variety of systems, on the condition that the dynamics are known, and that the energy level increases or decreases continuously towards the goal state.

In independent work, Boone presented a control method based on graph search [Boo97a]. He divided the problem's state space into $20 \times 20 \times 20 \times 20$ (160,000) tiles; Boone then used the tiles as nodes of a graph, and progressively generated edges from a combination of action and known acrobot dynamics. i.e. an edge was generated each time the search algorithm required knowing which tile/node the system state would be in after taking a control action from another tile/node. The target of the search was the maximisation of the acrobot's tip height.

In the same paper, Boone also combined this method with an online learning model of the controlled system's dynamics to replace pre-existing knowledge of the controlled system's dynamics. This method assumes that the input at each timestep represents the full state of the controlled system and likely requires minimal noise in the dynamics of the system, but was able to drastically decrease the number of actual control steps until a successful control strategy was found. This was done by running the graph search mentioned above using the online model to simulate the dynamics.

Interestingly, this approach could likely also be applied in conjunction with the control methods introduced in this work, and effect a similar reduction in the number of effective control attempts needed until success. This would particularly useful when applying these methods directly to learning to control a real-world system (e.g. controlling an actual, physical, cart-and-pole system as opposed to a simulated version of it), for which the running of a large number of control attempts is particularly inconvenient or may damage the controlled the system. The control system resulting from the combination of Boone's online learning method and the evolutionary generation of controllers would in effect behave like a rapidly adapting 'black box' controller.

A popular variant of the acrobot and associated solutions

Another variant of the acrobot system is also popular; in which the bottom link's length is doubled, while keeping an unchanged mass, and different moments of inertia are used [YNTI05]. The torque values applied by the controller are also real (not bang-bang). The fact that this variant of the acrobot can be swung up with a single back and forth swing (see ref [YNTI05], Figure 5) seems to indicate that this version of the acrobot poses less of a challenge. Several switching controllers were presented using this problem, varying in the swing-up and switching method used, but all using a Linear-quadratic

Regulator (LQR) derived from the system's dynamics to maintain the acrobot in the handstand position once swing-up is achieved.

Yoshimoto et al. combine multiple traditional linear controllers derived from the dynamics of the system, and a RL method based on the SARSA algorithm to choose which of these controller should be used at any given state of the acrobot [YNTI05].

Kawada et al. [KFOY05] evolve a swing-up trajectory (i.e. a series of actions, not a controller) via a genetic algorithm; a complex fitness function ensures the evolved trajectories are both short and suitable to be taken over by the stabilising controller.

Duong et al. [DKUY08] evolved the weights of a neural network (NN) to control the acrobot during the swing-up phase. The NN had four neurons in the hidden layer, and one output neuron which produced the torqued to be applied to the acrobot joint.

Reinforcement learning solutions

Most notably, Sutton [Sut96] set the swing-up problem as bringing the tip (the feet) of the acrobot above one link height over the bar; bang-zero-bang control actions, and a control frequency of 5Hz were used. Sutton combined the SARSA algorithm with a coarse function estimator which divided the state-space of the problem according to an elaborate scheme, using 48 separate tilings, resulting in 18,648 tiles. At each timestep, the input state was translated in a combination of tiles on which the SARSA algorithm was applied. No explicit pre-existing knowledge of the dynamics was used, but the tiling of the input state used was very specific to the problem. However the sparsity of reinforcement given (-1 for each timestep not in a successful state, until success), makes this work impressive. It should be noted that contrary to the methods which are the subject of this thesis, this approach is limited to Markovian problems where the full state of the system controlled is given at each timestep.

Da Motta Salles Barreto and Anderson introduced the harder version of the acrobot studied in this thesis [dMSBA08]. It differs from Sutton's by quadrupling the frequency of control actions, from 5Hz to 20Hz. Da Motta Salles Barreto and Anderson find this makes the acrobot swing-up problem significantly harder, and though they successfully apply the reinforcement learning methodology presented and a policy iteration method [LP03], they were unable to generate a successful controller with any of the several evolutionary approaches tried. These difficulties make this challenging version of the acrobot swing-up problem particularly interesting.

In summary, the acrobot swing-up problem was found hard enough to generally require the injection of large amounts of knowledge of the system into the controller. The approach in this thesis of limiting to a minimum the information about the controlled system's dynamics given to the controller, and giving only sparse feedback, make it harder. Furthermore, the version of the problem used in this thesis will be the more difficult version introduced by da Motta Salles Barreto and Anderson, for which they were unable to find any evolutionary solution.

2.1.4 The mountain car problem

The mountain car problem is a standard reinforcement learning problem in which a simple simulated car must climb to the top of a mountain from the valley (See Figure 2.6). It was initially introduced by Moore [Moo91] to illustrate a control method combining environment modelling and long-term policy optimisation on a real-valued problem. The problem became more widely used after its inclusion in Sutton and Barto's introductory reinforcement learning book [SB98].

The problem's difficulty arises from the limitation of the car's engine which is not powerful enough to propel it to the top of the mountain from a cold start at the bottom of the valley. The only possible method for the car to reach the top of the mountain is to first swing back and forth to accumulate enough speed to be able to climb the mountain; consequently a successful control must move the car away from its final target before reaching it.

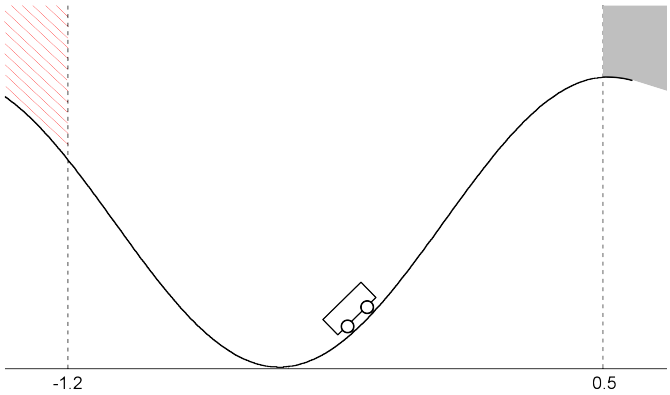


Figure 2.6: The mountain car problem. The car must reach the greyed area to the right; additionally, if it reaches the red zone to the left it is blocked from going further up and its velocity is reset to zero.

The equations of motion and experimental settings used for the mountain car problem in this work are taken from Sutton and Barto's book [SB98]:

$$x_{t+1} = bound\left[x_t + \dot{x}_{t+1}\right]$$

$$\dot{x}_{t+1} = bound\left[\dot{x}_t + 0.001a_t + 0.0025\cos 3x_t\right]$$

where x_t and $\dot{x}_t$ are respectively the car's horizontal position and velocity at time t, and a_t is the output of the controller at time t which must be one of $\{1, -1, 0\}$, corresponding respectively to the car's engine applying force to the car forwards (towards the right), backwards (towards the left), or not applying any force to the car. The $bound$ function keeps x within the range $[-1.2, 0.5]$, and $\dot{x}$ within the range $[-0.07, 0.07]$. When the car's position x reaches its left bound, the car's velocity $\dot{x}$ is reset to zero. For each controller evaluation, the initial position and velocity of the car are random and taken uniformly from these ranges. A controller receives a negative reinforcement of -1 for each timestep before success, and 0 on success. The control run is successful and stopped if the car reaches the target position of 0.5. A control run is limited to a maximum of a thousand iterations, and the inputs to the controller — the position and velocity, x and $\dot{x}$ — are both normalised to the range $[0, 1]$.

Previous solutions

Moore's initial solution of the problem was obtained via a control method generating a model of the environment from the effects of the controller's actions; immediate actions were then selected based on their long-term consequences in simulated runs relying on the generated model of the environment [Moo91]. Sutton applied the reinforcement learning method SARSA, combined with coarse inputs to solve a variety of control problems, amongst which the mountain car problem [Sut96].

More recently, Metzen et al. used the mountain car problem as an online problem, effectively allowing multiple controller to be tested during each control run, and stopping a control run only on success [MEKK08]. They obtained good results with a neuroevolutionary method, despite these methods usually requiring a separate control run for each controller [MEKK08]. Da Motta Salles Barreto and Anderson used the mountain car problem as one of several problems to demonstrate the competitiveness against other recent methods of a reinforcement learning method they introduce [dMSBA08].

In summary, the mountain car task is a classical reinforcement learning problem which has been solved with a variety of methods (though not using any GRN-based method). Beyond generating the first successful controller for this problem via a GRN-based method, this problem is of interest in this work as it puts forward the ability of the method tested to produce controllers that generalise well. This is due to the problem's experimental setup, which requires each controller to start from a random position in the state-space (both the initial position and velocity of the car being randomly selected from the full range of possible values).

2.2 Natural gene regulatory networks (GRNs)

Our understanding of the workings of GRNs has been rapidly increasing in recent years. This section summarises some of our current knowledge on the subject. Most relevant to the work in this thesis is the regulatory aspect first described. This work focuses on models of GRN in isolation with few genes; the details of the network structure, network motifs, and self organisation displayed by natural GRNs are less important.

2.2.1 Regulation

GRNs in effect act as cell controllers, and are composed of the set of interactions between genes and proteins in a cell; activated genes produce proteins which can in turn regulate the activation of other genes [Dav06]. These regulatory proteins (known as transcription factors) can also interact in a wide range of ways to determine the activation or, to the contrary, repression (non-activation) of genes. Various other regulatory mechanisms exist, but play a significantly lesser role in regulation and will not be considered in this work. The effect a protein has on a gene's activation can take many forms; it can individually promote or repress the activation of the gene, but proteins can also combine, controlling the activation of a gene via the logical equivalent of AND or OR logical functions. The presence of a specific protein can also increase or decrease the effect of another protein on the gene's activation [YBD01].

2.2.2 Network structure

Interestingly, both GRN and networks of protein interactions display scale-free topologies [BO04] [BLA+04]. The topologies of scale-free networks are such that the connectivity of nodes follows a power law distribution [BA99]: the frequency of the nodes decreases at a higher rate than their connectivity. Networks with scale-free topologies are robust [BO04]. Scale-free topologies can arise and be maintained through the combination of the two generic mechanisms of growth and preferential attachment : the network expands through the addition of new nodes, which attach preferentially to previously well-connected nodes [BA99]. And indeed, GRN evolutionary genotypic growth occurs mainly through gene duplication [TB04], similarly, the scale-free topology of protein interaction networks is likely to be the result of preferential attachment [EL03]. Additionally, GRNs have also been shown to be hierarchical [ED09] and modular [BLA+04]. All these features illustrate the importance of evolutionary dynamics in the generation of natural GRNs.

2.2.3 Network motifs

At a local level, GRNs display certain patterns of interconnections (termed 'Network Motifs') in frequencies much higher than would be expected in randomised networks [SOMMA02]. The prevalence of these motifs in GRNs is not directly attributable to gene duplications [TB04], and has been shown to be the subject of convergent evolution in *E. coli* and yeast [CW03]. This suggests that these motifs are evolutionary desirable, and are not artifacts of the process of evolutionary growth of the GRNs.

2.2.4 Self organisation

Through cells, natural GRNs display an impressive ability to self-organise. Examples of GRN-controlled cells self-organising include:

- Self-organising bacteria colonies, exhibiting rich, adaptive, behaviours through local sensing and communication [BJ03].
- Some cellular slime molds form temporary bodies, aggregating into migrating multicellular slugs [MH01].
- Multicellular eukaryotic bodies, with a single genotype, develop from a single cell to trillions of cells. Cellular differentiation, through the specialisation of gene activation patterns, plays an important part in that process [DRO+02].

2.3 Gene regulatory network (GRN) models

In this section, evolutionary GRN models (models conceived to be evolvable via artificial evolution methods such as genetic algorithms (GAs)) will be reviewed. In a rough chronological order following the research trends, this review will begin with Kauffman's pioneering work on random boolean networks (RBNs), followed by neural network based GRN models. The more complex developmental GRN models aiming to generate complex shapes and designs will then be reviewed before looking at the most recent direction, GRN models for control. GRN-based control being the topic of this thesis, particular

attention will be given to the two models most used in this area : Banzhaf's artificial regulatory network (ARN), and Bentley's fractal GRN (FGRN).

2.3.1 Random boolean networks (RBNs)

In 1969, Kauffman introduced random boolean networks (RBNs) [Kau69] as a way to model what he called "genetic nets". In RBNs each gene is a boolean variable which is set to true when activated, and false otherwise. As in cellular automata, the array of genes is iteratively updated, each gene taking a new value which is function of the state of a subset of the other genes in the previous iteration. As opposed to a cellular automaton, the interactions are not locally restricted, the subset of input genes influencing any given gene being randomly selected.

Formally, RBNs are composed of a set of N boolean variables each associated with a boolean function of K inputs randomly taken from the N variables. All N variables are updated simultaneously and take as value the current result of their associated function [Kau93]. Depending on the number K of input genes, the resulting activation patterns have distinctive appearances. Kauffman distinguished three regimes in which a RBN can be:

- $K > 2$**.** Chaotic regime.
- $K = 2$**.** Phase transition from order to chaos.
- $K = 1$**.** Ordered regime.

Figure 2.7 shows typical activation patterns for various value of K.

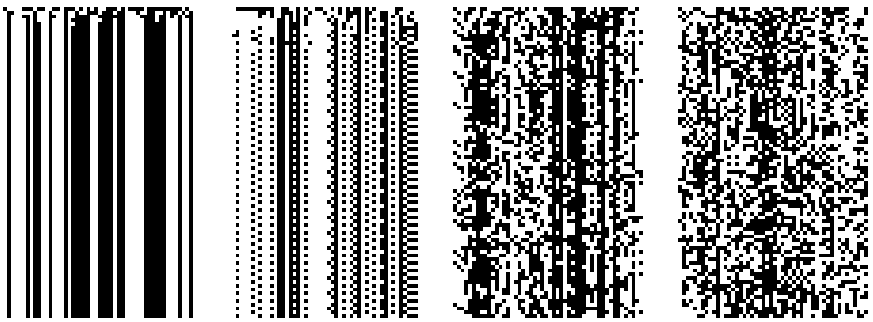


Figure 2.7: Typical activation patterns of RBNs through time for 50 genes with, from left to right, K = 1, K = 2, K = 3, and K = 4.

RBNs with low Ks can only transition ultimately from each state to a limited number of other states. These forms basins of attractions, repeating indefinitely the same activation patterns which can be seen as roughly similar to cell types [Wue98].

Kauffman also hypothesised natural GRNs operate in the phase transition from order to chaos and several predictions were made about the characteristics of biological GRNs based on the idea that RBNs with low K values were an accurate biological model. The number of genes in the human genome was initially predicted to be 2,000,000 [Kau69], then 100,000 [Kau93], far off the current estimation around 30,000 [Dav06]. Other predictions associating RBN attractor basins with biological cell types were also incorrect.

Random boolean networks were the first evolutionary model of GRNs and displayed interesting dynamics, but have had little practical applications.

2.3.2 Neural network based GRN models

Though inspired by a different natural system, neural networks have been used as evolutionary GRN models.

Mjolsness et al. developed a complex developmental system based on a combination of neural networks and a L-System [Lin68a] [Lin68b]-like grammar, for the purpose of modelling biological systems [MSR91].

A discrete-time recurrent neural network (DTRNN) [CSSM89] was used as a GRN to accurately model the activation patterns in the cells of a *C. elegans* egg after the first four cell divisions [GW03]. Though the activation patterns were successfully reproduced, it is unlikely the evolved network was an accurate model of the biological GRN studied.

Wagner presented a GRN model [Wag94] in effect very similar to a DTRNN, the only difference between the presented GRN model and a DTRNN being the absence from his model of a bias input. The model was used initially to study evolutionary dynamics, first looking at the effect of gene duplication on gene activation patterns, then studying the evolution of evolutionary plasticity [Wag96], showing that, among solutions of equally high fitness, evolution favours solutions of higher phenotypic robustness.

These models, though exploited to provide interesting insights into evolutionary dynamics, are of limited interest for control applications, lacking features of natural GRNs such as regime-switching and complex protein interactions.

2.3.3 Developmental GRN models

One of the most impressive products of natural GRNs being the generation of large multicellular organisms composed of myriads of cells with many different functions, it is not surprising that GRN models would be created with the aim of providing a platform for the automatic generation of complex shapes and designs through artificial evolution. A less obvious application was the use of developmental GRN models for the generation of controllers (e.g. neural network controllers); this is different from the subject of this thesis which aims to rely on GRN models to directly control a system.

Jakobi presented a strongly biologically-inspired GRN model based on a DNA- network (ANN) robot controllers [Jak95]. Reil introduced a similar, simpler model, titled “artificial genome” (AG) and analysed its gene activation dynamics [Rei99]. Similarly to RBNs Reil found attractors and ordered, chaotic and complex regimes; robustness of gene expression patterns to disturbances was also displayed. Working with Reil’s AG model, Hallinan and Wiles studied the effects of changing the synchronicity of the update rules AG [HW04b] [HW04a]. Hallinan and Jackway then studied the proportion of network motives in evolved and random networks, with inconclusive results [HJ05].

Eggenberger introduced the artificial genetic regulatory system (AGRS), a multicellular model based on digital strings for the evolutionary development of neural networks [Egg97] [Egg01]. This model was used for pattern formation through the introduction of morphogen gradients [ED99], before, coupling it with simulated physics, producing a variety of shapes through morphogen-

esis [EH03] [EH04a]. The AGRS was then applied to the developmental generation of lens shapes [EH04b].

Bongard created a GRN model for the evolution of designs for multicellular robots in a virtual, environment with simulated physics [Bon02]; the GRNs directed the development of both the morphology and control system of the robots, with impressive results.

Kumar introduced the evolutionary developmental system (EDS) [KB03] combining a simple GRN model in effect similar to a neural network, with complex cell physics, to evolve simple multi-cellular shapes. The physical properties of the system were evolved in conjunction with the GRNs. This model also had a limited control application, being used for very basic obstacle avoidance [Kum05].

Mattiussi introduced analog genetic encoding (AGE), a GRN model based on a DNA-like genome [Mat05] from which free-floating components were extracted, connecting to form networks, which were then evaluated as electrical circuits or neural networks [MF07]. Notably, this developmental method of generating neural network controllers was successfully applied to the double pole balancing problem with velocities withheld [DMF06].

A wide variety of interesting approaches have been taken to create GRN models for development, these models are not however suitable for direct control without modifications.

2.3.4 GRN models for control

Despite the role of natural GRNs as a cell's controller, GRN models have known comparatively little use for control. This section does not cover the most popular such models, the ARN and the FGRN, which are instead discussed in the following sections.

Quick introduced a simple GRN model for control which was applied to basic temperature control and mobile light-following problems [QNDR03]. Knabe et al. extended Quick's model to evolve GRNs able to produce a periodic signal [KNSQ06] [KNS06]; the resulting model was also use for a simple developmental application [KSN08].

Those models were only applied to very simple control problems, and have not been tested to the same extent as the two models below.

2.3.5 Artificial regulatory networks (ARNs)

Banzhaf introduced the artificial regulatory network (ARN) [Ban03b], a GRN model based on DNA-like binary strings, the genes being extracted from the initially random binary strings from any location exhibiting a given promoter pattern. Protein matching to genes' regulatory sites is determined by the product of a XOR operation. The system was shown to exhibit interesting dynamics similar to those seen in natural GRNs, notably oscillatory behaviours and the ability of genes to be expressed at different rates [Ban03a].

Kuo et al. showed ARNs could be evolved to match target output functions such as sinusoids, exponentials and sigmoids [KLB04]. Kuo et al. then produced scale-free ARN topologies — a common characteristic of biological networks — through gene duplication and divergence [KB04]; however this was done through biologically implausible whole-genome duplication events, the ability of the model to

keep a scale-free topology through incremental duplication/divergence events would be more desirable for real world applications. Similarly, Leier et al. studied the proportions of network motifs in ARNs obtained through duplication and divergence [LKB07].

More recently, and concurrent with the work presented in this thesis, Nicolau et al. extended the ARN model with mechanisms for input and output and have successfully applied it to the single pole balancing problem [NSB10]. Nicolau et al. then applied the system to algorithmic index trading [NOB12], obtaining performances similar to a grammatical evolution system on the same data. Murphy et al. extended the model further with the addition of a grammar increasing the potential expressivity of the system's output [MNH$^+$12], without detriment to the system's performance on the pole balancing problem. Lopes and Costa extended the ARN model by adding as an additional step extraction of a simplified network [LC11] and successfully applied the resulting system to pole balancing.

2.3.6 The fractal GRN (FGRN) model

Bentley introduced the FGRN [Ben04b] (as "fractal proteins"), a GRN model in which the genome consists of a list of genes with different functions, and the complex interactions between proteins and genes are replaced by mathematical operations between two-dimensional fragments of a fractal. The FGRN model was initially applied to developmental problems : FGRNs were evolved to produce specific gene activation patterns [Ben04b]. FGRNs were also evolved to approximate from an input the square root function [Ben05]. FGRNs were further shown to be fault-tolerant [Ben05], and able to increase in robustness when left to evolve after the maximal fitness was reached [Ben04a].

The first control application of the FGRN model was the evolution of a controller for guidance of a robot to a fixed destination while avoiding walls in the way [Ben03a]. Zahadat et al. then successfully applied the FGRN model on a grid-world robot box-pushing problem [ZK08].

Concurrent with the work in this thesis, Zahadat et al. then applied the FGRN model to the distributed control of a modular robot for locomotion [ZCS$^+$10] [ZSC12], and presenting a method to translate an evolved FGRN into a simpler algorithmic representation [ZS12]. A more detailled version of Zahadat's work is available in his thesis [Zah11].

The recent increase in the number of publications on the subject of GRN-based control indicates increasing interest in the field. The FGRN is chosen as the starting point for this work, as it has had a more diverse range of applications; it will be described in detail in the next chapter.

Chapter 3

Fractal Gene Regulatory Networks (FGRNs) for Development

In this chapter, Bentley's original FGRN model will be described in detail, before covering some preliminary experiments; then, to ensure the suitability of the FGRN model for control, the evolvability of the FGRN model is first tested on a developmental problem. Developmental use tests whether the FGRN is able to generate complex output patterns from only its internal workings, without the help of any external input.

Different ways of using the system's output to produce a phenotype (output-to-phenotype mapping strategies) are also tested. The FGRN system can produce two different types of output: *binary* from the behavioural (output) genes' activation state, and *real* which is broadly equivalent to what would be the concentration of proteins produced by those same behavioural genes. Two output-to-phenotype mapping strategies, each based on one of these output types, are tested. It is shown that the more dilute approach, using the FGRN system's *real* outputs to influence the phenotype in multiple ways, can be advantageous. It is foreseen that this could be of use when applying the system to harder control problems.

For this developmental task an irregular phenotype is needed: the mathematical constant π is chosen as it is notably irregular, and the irrational number most widely known as such. Each output-to-phenotype mapping strategy aims to reach π differently:

- Using the *binary* output, the binary representation of π is targeted as a gene activation series.
- Using the *real* output, an iterative process arithmetically combines the outputs of the system to target the value of π.

The precision obtained with the second method is impressive: initially, the precision of the result data type (sixty-four bit double precision floating point) is reached, forcing the use of non-native, higher precision data types to keep track of the system's performance (notably the precision of the system's parameters is not increased).

3.1 System description

In this section, the original FGRN model is described. FGRN genomes are used as both a medium for evolution (they are subject to mutation and crossover operations), and as running, developmental systems. A running FGRN genome takes no input and iteratively generates a series of outputs. This output series defines the phenotype for a given genome. Such systems are a form of artificial embryogeny, the phenotype being *developed* from the genome.

Therefore, as in biology, the mapping between genotype and fitness is indirect. The fitness of an individual genome is a function of its phenotype only. This allows multiple genomes to map to the same phenotype, a fact that can be used by artificial evolution to tune the genomes further than maximising the fitness, by in effect evolving evolvability itself [Wag96]. One of the beneficial outcomes of this is that more robust (less susceptible to external perturbations) phenotypes can be obtained by running the evolutionary process further than the point at which a maximal fitness is obtained [Ben04a].

The FGRN genome will first be presented; the dynamics of the running FGRN will then be explained, before detailing the fractal chemistry which is at the heart of the system.

3.1.1 FGRN genome

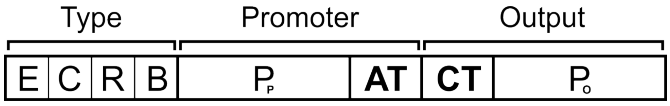


Figure 3.1: The composition of a FGRN gene. From left to right: the type field; the promoter section composed of the promoter protein definition P_p and the affinity threshold (AT); the output section composed of the concentration threshold (CT) and the output protein definition P_o.

The FGRN genome is a list of genes with different roles in the running system. All genes are composed of the same fields, but act differently depending on the type(s) they possess (see Figure 3.2). Each FGRN gene is composed of the following fields (illustrated in Figure 3.1):

- **Type**: The role of the gene in the running system. A gene can have any combination of the four types: environmental (**E**), receptor (**C**), regulatory (**R**), and behavioural (**B**). In the running system, during development, a gene with multiple types acts as multiple genes, each of one of its types, and otherwise identical. The role of each gene type will be described in detail below.
- **Promoter protein definition** (P_p): defines the promoter protein. The promoter protein, combined with the Affinity Threshold (AT), controls the activation of the gene. The concept of a promoter *protein* departs from biology, as a biological gene's promoter is not a protein (which is the product of a gene), but a section of the gene's DNA.
- **Affinity threshold (AT)**: controls gene activation.
- **Concentration threshold (CT)**: controls the amount of output protein produced when the gene is activated. Initially this was also used as part of the promoter [Ben04b], but in Bentley's further work [Ben05], the activation condition based on CT was relaxed.
- **Output protein definition** (P_o): defines the output protein, which is produced in variable concentration when the gene is activated.

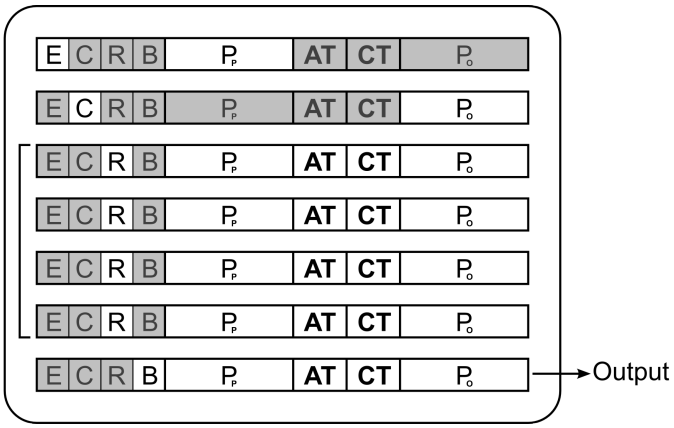


Figure 3.2: A FGRN genome with one environmental (**E**) gene, one receptor (**C**) gene, four regulatory (**R**) genes, and one behavioural (**B**) gene. All genes have the same structure and are composed of the same fields. The greyed out portion of the genes are not used: receptor genes do not use the promoter protein definition P_p, and environmental genes do not use the output protein definition P_o; neither use the affinity or concentration thresholds (AT and CT). The behavioural gene produces a boolean or scalar output.

Gene types

A gene can have any combination of the four gene types below (the role of each gene type is illustrated Figure 3.3), all genes are subject to evolution:

- **Environmental** genes allow for some constant protein input into the cytoplasm. The environmental promoter protein is always added (through the receptor protein) to the cytoplasm, at saturation concentration.
- A **receptor** gene's output protein acts as a mask for environmental proteins, letting through only part of them. This allows a part of the cytoplasm to be reserved for internal computation only. A genome can contain an arbitrary number of receptor genes, but only the first one is considered. If no receptor gene is present, environmental proteins are allowed fully into the cytoplasm.

  Having a separate receptor gene is particularly useful for applications which require several environmental genes, such as the control applications studied in further chapters.
- **Regulatory** genes only directly affect the cytoplasm, acting therefore as hidden processing units.
- Each **behavioural** gene acts as a system output.

Genome evolutionary features

The following gene-based mutations are used; all mutation values are taken from a uniform distribution:

- Type: a type is added or removed from the gene.
- Affinity and concentration thresholds: a random value is added to the threshold.

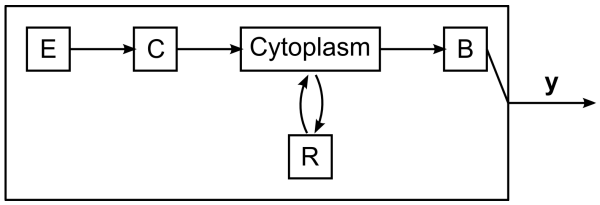


Figure 3.3: A graph of the roles of each gene in a running FGRN. Environmental(**E**) genes are masked by the receptor(**C**) gene before being added to the cytoplasm. Regulatory(**R**) genes are activated as a function of the cytoplasm and output back into it. Behavioural(**B**) genes produce the system's output.

- Promoter and output protein definition P_p and P_o initialisation and mutations mechanisms are distinct and will be covered in the description of the protein chemistry (see Section 3.1.4).

The initialisation and mutation ranges used for each gene field are included in Table 3.1. The following genome-based mutations are used:

- Gene duplication: a random gene is duplicated and the duplicate appended at the end of the genome.
- Gene deletion: a random gene is removed.

Crossover occurs down to the gene level and is independent of gene types; for each gene position in the genome, for each parameter in this gene (e.g. gene type, thresholds, protein coordinates), the value used in the child genome is randomly selected from one of the two parents. If the crossed-over genomes are of different sizes, the size of the genome produced will be randomly selected from one of the two parents. Bentley used a mutation rate of 0.01 per gene component for similarly sized genomes [Ben04b]; for preliminary runs of the developmental experiments detailed in the next two sections of this chapter, it was found that the system generated fitter final solutions with a mutation rate of 0.02. A mutation rate of 0.01 was therefore applied from there on. Crossover is always applied.

Before the FGRN can be used to produce an output, there is a transition step from the evolved genome to the running genome:

- Genes with multiple types are transformed into multiple genes of a single type.
- Only the first receptor gene in the genome is kept.

Table 3.1: The initialisation and mutation ranges for the FGRN gene fields.

Field	**Initialisation range**	**Mutation range**
Affinity threshold	$[-10,000, 10,000]$	$[-5,000, 5,000]$
Concentration threshold	$[0, 200]$	$[-100, 100]$

3.1.2 The FGRN genetic algorithm (FGA).

The use of a GA as optimiser in conjunction with the FGRN genetic representation, or indeed any artificial GRN-based representation, is consistent as evolution is the method used by nature to produce fit GRNs. The FGRN model is also particularly suited to GAs (as opposed to optimisation methods relying on solutions having a fixed number of parameters), allowing, as in nature, the duplication or deletion of whole genes. The GA used in conjunction with the FGRN model in all published work prior to this one was created by Bentley [Ben96]; as it is nameless it will be abbreviated here to FGA.

The FGA preserves a large portion of the population from one generation to the next: the fittest 20% of the previous generation are kept in the population unmodified. The remaining 80% is filled with child genomes produced from two parents randomly selected from the fittest 40% of genomes, then mutated. Genomes are aged at each generation and can only persist unmodified in the population until they reach a maximum age of ten. The main loop of the FGA is detailed in Algorithm 3.1; the constants used and the values they generally, and here, take are shown in Table 3.2. The FGA strongly favours the fittest individuals as parents. This extreme elitism is likely to lead to premature convergence to only locally optimal solutions.

Algorithm 3.1: The FGA main loop being run for GenerationCount generations. The functions used in the main loop are detailed in the Appendix in Listings A.1, A.2, and A.3.

```
declare integer CarriedOverCount := PopulationSize − ChildrenCount
declare array children := array(ChildrenCount)
declare struct child_genome
declare array population := newRandomPopulation(PopulationSize)
sortByDecreasingFitness(population)

{ Run for GenerationCount }
for generation := 1 in GenerationCount
  { Generate children }
  for i := 1 in ChildrenCount
    parent1 := pickParentGenome(population)
    parent2 := pickParentGenome(population)
    child_genome := crossover(parent1, parent2)
    mutate(child_genome)
    children[i] := child_genome
  end

  ageAndRemoveExpired(population)

  { Keep the top pre−existing genomes, and add the children to the population }
  population := population[1..CarriedOverCount] + children
  sortByDecreasingFitness(population)
end

{ The fittest genome obtained }
return population[1]
```

Table 3.2: The constants associated with the FGA, with their usual values.

Constant	Description	Value
PopulationSize	Number of genomes in the population	100
ChildrenCount	Number of genomes generated in a generation	80
MaximumAge	Maximum age of an individual	10
ParentCoefficient	Proportion of the population used as parents	0.4
RandomParentCoefficient	Chance of picking up a parent randomly from the whole population	0.01

3.1.3 Fractal proteins

In a natural GRN, the gene/protein/environment interactions are many and complex. FGRN restricts itself to those that are hoped to be the main ones [Ben09]:

- The mapping of a gene's protein coding section to the protein product.
- The interactions amongst proteins floating in the cytoplasm.
- The interactions between those proteins and the promoter section of individual genes, defining the activation of these genes.

Bentley chose to not attempt to replicate these complex biological processes, but to instead substitute a relatively computationally faster set of interactions, following a principle of "deliberately incorrect modelling" [Ben09]. The set of interactions defined by Bentley, fractal chemistry, is based on the concept of the protein as a two-dimensional sampling of the Mandelbrot fractal, and on simple mathematical operations. As seen in Figure 3.1, each FGRN gene contains two protein definitions: P_p for the promoter protein, and P_o for the output protein. Each of these is a triplet of real numbers $< x, y, z >$.

Defining a gene promoter as a protein is an odd choice given the completely different functions in nature of a gene promoter and its protein encoding section, and is not justified in Bentley's work.

A fractal protein defined by $< x, y, z >$ is a square subset of the Mandelbrot set with sides of length z and centre coordinates $< x, y >$ that define the real and imaginary parts of a complex number [Ben03b]. The colouring from white to black of a sample point within the square represents the speed with which the value at that point falls out of the range $[0, 2]$ upon iteration of the Mandelbrot equation, with black sampling points generating values that remain bounded within a radius of 2. In principle a fractal protein is an object of arbitrarily high complexity, though in practice to limit computational demands the square subsets are usually implemented as 15×15 bitmaps, with the $< x, y >$ value of a pixel being that of the point at its centre. It might be questioned at this point whether a 15×15 bitmap of a coarsely represented fractal is sufficient to capture the complexity of a fractal.

A fractal protein has an associated concentration that alongside the concentrations of other protein constituents determines the degree to which it can act towards further gene activations. Although a protein's concentration is stored as a single real value (in the range $[0, 200]$ for historical reasons, 200 being the saturation value arbitrarily chosen by Bentley [Ben04b]), it can also be represented as a bitmap for

ease of use in the merging and comparison chemistry operations described below; indeed, the cytoplasm, merger of multiple proteins, also merges the protein's associated concentrations, resulting in a patchwork of concentrations being associated with it. Figure 3.4 shows an example of a fractal protein at 15×15 resolution and its associated concentration bitmap. Each point of a concentration bitmap is coloured from white (absence of protein) to full red (protein present in saturation).

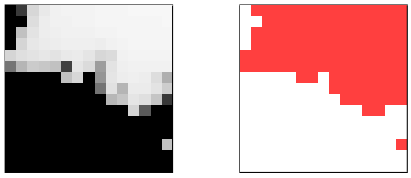


Figure 3.4: A fractal protein (left) and associated concentration bitmap (right). Note that the non-white points of the concentration bitmap correspond to the non-black points of the protein. The closer to saturation the concentration of a protein is, the redder its color part is (a low concentration would be a faded pink).

As in ref [Ben04b] and all subsequent FGRN work, when evolving FGRNs, the promoter and output protein coordinates of the genes of the initial genomes are randomly assigned the coordinates of one of ten pre-evolved proteins (see Figure 3.5), the use of this set of proteins was found by Bentley to reduce the number of evaluations to successful solution in his initial work on simple pattern generation [Ben04b]. Each coordinate x, y, or z is mutated independently. A uniformly distributed random real value in the range $[-0.5, 0.5]$ is added to the mutated coordinate. When two protein definitions are crossed over, as part of a gene crossover, each coordinate of the resulting protein definition is randomly chosen from either gene's corresponding protein definition. Figure 3.6 gives an overall view of an FGRN gene, including the link from protein definition to protein.

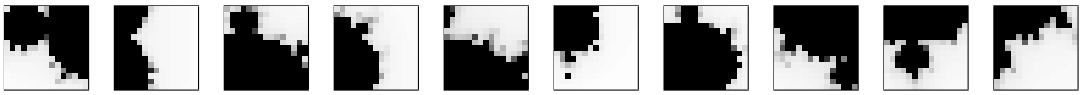


Figure 3.5: The ten pre-evolved fractal proteins used in FGRN genome initialisation. This protein set was generated by Bentley to have proteins with a varied set of boundaries [Ben04b]. Genetic drift tends to produce all black or all white proteins of less interest; in particular this can be seen in the output proteins of environmental genes, and the promoter proteins of receptor genes, which are not subject to direct evolutionary pressure, as they are not a functional part of the running system. This can be seen in the genome illustrated in Figure 3.13 at this end of this chapter.

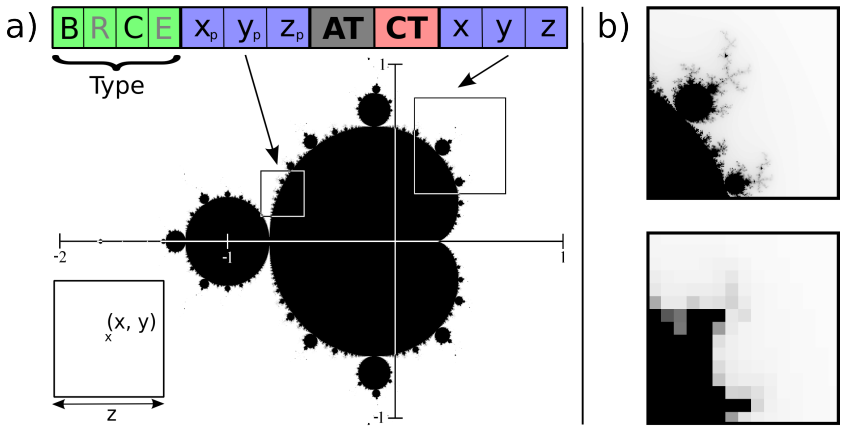


Figure 3.6: A gene in detail. a) The type specifies that this gene is expressed as both a behavioural (**B**)and a receptor (**C**) gene, but not as a regulatory (**R**) or an environmental (**E**) gene; the promoter $< x_p, y_p, z_p >$ and output $< x, y, z >$ protein coordinates define different fractal portions of the Mandelbrot set. Bottom left: a schema describing the mapping from protein coordinates to fractal portion. b) Top: the fractal portion pointed at by the output coordinates $< x, y, z >$ of the gene. Bottom: the resulting 15×15 fractal protein.

3.1.4 Protein chemistry

The protein chemistry operations underpinning the working of an FGRN: decay, mask, merge, and compare, are detailed below.

Decay

At the end of each running iteration the concentration values of the regulatory proteins present in the cell are decayed and those with values less than a fixed threshold ϵ_d have their concentration set to zero (and so are no longer considered present in the cell). This process is illustrated in Figure 3.7a, and formalised in Equation 3.1.

$$c_{i+1} = \max(0, \lambda_p \cdot c_i - \epsilon_d) \tag{3.1}$$

where:

- c_i is the protein concentration at iteration i.
- λ_p is the persistence coefficient (set to 0.8).
- ϵ_d is the minimum protein diffusion (set to 0.2).

Mask

This is the filtering mechanism by which a receptor gene controls the parts of environmental proteins that enter the cytoplasm. Full black regions of the receptor protein bitmap are treated as opaque and all others as transparent. An example of masking is shown in Figure 3.7b.

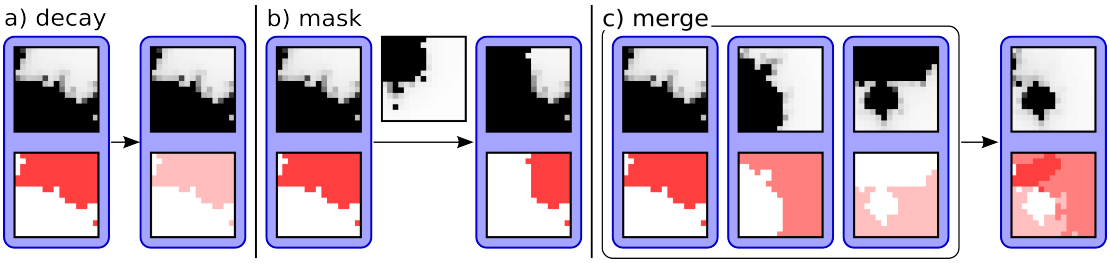


Figure 3.7: Protein chemistry. a) Decay: the concentration associated with a protein is reduced. b) Mask: a protein (centre, above arrow) masks another. c) Merge: in this case three proteins and their associated concentration bitmaps are merged into one. Note the patchwork nature of the merged protein's concentration bitmap.

Merge

In principle a merged product is calculated by iterating through the fractal equations for each protein and choosing as winner for each pixel that value that becomes unbounded most quickly. In practice merging can be carried out more simply by comparing the stored values for the bitmaps and choosing at each point the maximum pixel value (pictorially, that closest to white); merged proteins thus tend to be dominated by white regions, as can be seen in Figure 3.7c. When proteins are merged the concentration at each point becomes that of the winner, producing a concentration bitmap with the 'patchwork' appearance in this example.

Compare

The promoter protein bitmap is compared with the merged product, and the sum of absolute differences between its non-black pixels and the corresponding pixels in the merged product is calculated, which determines the probability of activation of regulatory and behavioural genes, as detailed in Equation 3.2

$$p_{a\,i,j} = \begin{cases} 0.5 + (tanh(\frac{\Delta P_{i,j} - AT_i - C_t}{C_s}))/2.0 & \text{if } AT_i \geq 0 \\ 0.5 - (tanh(\frac{\Delta P_{i,j} + AT_i - C_t}{C_s}))/2.0 & \text{if } AT_i < 0 \end{cases} \tag{3.2}$$

where:

- $p_{a\,i,j}$ is the activation probability of gene i at iteration j.
- $\Delta P_{i,j}$ is the sum of absolute differences between the non-black pixels of the promoter protein of gene i and the corresponding pixels in the merged product at iteration $j-1$.
- AT_i is the affinity threshold of gene i.
- C_t is a threshold constant (set to 0).
- C_s is a sharpness constant (set to 20).

The mean value $\bar{c}$ of the corresponding pixels in the merged concentration bitmap is calculated.

Gene activation

If a gene is determined to be activated, $\overline{c}$ is used in conjunction with the concentration threshold CT to determine the gene's output level.

For **regulatory genes**, protein output r is determined according to Equation 3.3. The output r is then added to the current concentration of the protein in the cell, which is then constrained to the range $[0, 200]$. The constants C_w and C_i are set to the same values as in previous FGRN experiments [Ben04b]. A notable characteristic of the protein output r not mentioned in Bentley's work is that r can be negative, in effect speeding up the disappearance of a protein from the cytoplasm.

$$r = \overline{c}\frac{tanh(\frac{\overline{c}-CT}{C_w})}{C_i} \tag{3.3}$$

where:

- r is the gene's protein output.
- $\overline{c}$ is the mean concentration.
- CT is the gene's concentration threshold.
- C_w is a constant (set to 30).
- C_i is a constant (set to 2).

For each **Behavioural gene**, the output o is determined according to equation 3.4.

$$o = sgn(AT) \cdot (\overline{c} - CT) \cdot x \tag{3.4}$$

where:

- o is the gene's output.
- AT is the gene's affinity threshold.
- $\overline{c}$ is the mean concentration.
- CT is the gene's concentration threshold.
- x is the first coordinate of the gene's output protein.

The system is run through a fixed number of iterations, each consisting of creating a new cytoplasm by merging the environmental proteins (masked with the receptor protein) and the output proteins of regulatory genes with a non-null concentration. Regulatory and behavioural genes are activated probabilistically as a function of promoter-cytoplasm interactions : an activation probability p_a is calculated for each gene, and the gene is only activated if a random value uniformly distributed on the range $[0, 1]$ is less than p_a. The aim of this indirect, probabilistic, mechanism is to generate a smoother fitness landscape when evolving FGRN genomes, by allowing a single genome to produce multiple, closely related phenotypes, and letting evolution control the degree of randomness in gene activation. In practice early FGRN genomes tend to display more variety in the phenotypes they generate, allowing for greater exploration, whereas when an optimal phenotype is reached, evolution typically reduces the randomness

of the genome that generated it, until there is no randomness left in the activations: once a good phenotype is found, evolution favours genomes that can reliably produce that phenotype and the activation probabilities produced by the associated running systems then tend strongly towards the values 0 or 1, thereby rendering the running systems purely deterministic. Furthermore, evolution then tends to favour genomes which are less sensitive to mutations in the activation patterns (phenotype) they produce; in effect evolution controls the evolvability of FGRN genomes.

3.1.5 FGRN dynamics

Algorithm 3.2 details the FGRN development process. The functions associated with Algorithm 3.2 are:

- **rand()**: get a uniformly distributed random real scalar in the range $[0, 1]$.
- **mergePromoterProteins(genes)**: merge the promoter proteins of an array of genes into a single protein.
- **merge(proteins_to_merge)**: merge an array of proteins with their associated concentrations into a new cytoplasm composed of a merged protein and a bitmap of the merged concentrations, as illustrated in Figure 3.7c.
- **decayConcentration(gene)**: decay a genes's concentration according to Equation 3.1.
- **maskProtein(protein, mask)**: return protein masked by the other protein mask as illustrated in Figure 3.7b.
- **compareCytoplasmToPromoter(cytoplasm, promoter)**: compare the cytoplasm (composed of merged protein and merged concentration bitmap) with a gene's promoter. Return a tuple composed of the sum of absolute differences ΔP, and the mean concentration $\overline{c}$.
- **activationProbability(AT, ΔP)**: calculate activation probability according to Equation 3.2.
- **regulatoryConcentrationUpdate(CT, $\overline{c}$)**: return the change in the concentration of the output protein of a regulatory gene according to Equation 3.3.
- **behaviouralOutputValue(gene, $\overline{c}$)**: get the output for a behavioural gene according to Equation 3.4.

Algorithm 3.2: The development main loop of Bentley's FGRN. Key: AT = affinity threshold, CT = concentration threshold.

```
{ split the genome into arrays of genes }
declare array environmental_genes := getEnvironmentalGenes(genome)
declare array behavioural_genes := getBehaviouralGenes(genome)
declare array regulatory_genes := getRegulatoryGenes(genome)
declare struct receptor_gene := getReceptorGenes(genome)[1]
declare struct receptor_protein := receptor_gene.output_protein

declare struct environmental_protein := mergePromoterProteins(environmental_genes)
declare struct default_protein := maskProtein(environmental_protein receptor_protein)

declare struct cytoplasm := <default_protein, saturation>

for i := 1 in developmentalStepCount
  { decay regulatory gene concentrations, merge into cytoplasm if still present }
  declare array proteins_to_merge := [<default_protein, saturation>]
  for each gene in regulatory_genes
    decayConcentration(gene)
    if gene.concentration > 0
      append(proteins_to_merge, <gene.output_protein, gene.concentration>)
    end
  end
  cytoplasm := merge(proteins_to_merge)

  { activate regulatory and behavioural genes }
  for each gene in regulatory_genes, behavioural_genes
    gene.activated := false
    <ΔP, c̄> := compareCytoplasmToPromoter(cytoplasm, gene.promoter_protein)
    p_a := activationProbability(gene.AT, ΔP)
    if rand() < p_a          { stochastic activation }
      if gene.is_regulatory
        gene.activated := true
        gene.concentration += regulatoryConcentrationUpdate(gene.CT, c̄)
      else if gene.is_behavioural
        if c̄ >= gene.CT
          gene.activated := true
          gene.output_value := behaviouralOutputValue(gene, c̄)
        else
          gene.output_value := 0
        end
      end
    end
  end

  outputs := [ ]
  for each gene in behavioural genes        { extract output values }
    append(outputs, gene.output_value)
  end
end
```

3.2 Preliminary experiments

The FGRN system originally developed by Bentley was fully reimplemented; this process allowed to solve some discrepancies in Bentley's published descriptions of the systems (e.g. the protein decay calculation [Ben04b], and the issues with free floating proteins described below), and to verify that the resulting system description, given in the previous section, was sufficiently descriptive to reimplement the system. Example proteins of known coordinates and aspect [Ben04b] were reproduced with the reimplemented system and were found to match exactly the proteins produced by Bentley's system. Bentley's previous experiments with the FGRN were reproduced with the reimplemented system (the results of these experiments were found to be consistent with those published by Bentley):

- A set of 10 distinct fractal proteins was evolved [Ben03b].
- FGRN genomes were evolved to produce specific activation patterns [Ben04b] (see Figures 3.8 and 3.9).
- FGRN genomes were evolved that output the square root of their input [Ben05].

B_1 : **+_+_+**
B_2 : **_+_+_**

B_1 : **+___+**
B_2 : **_+++_**

Figure 3.8: Target activation patterns lasting five developmental iterations. B_1 and B_2 are respectively the first and second behavioural genes of the FGRN. '**+**' signifies the gene must be activated at a given iteration, '_' that it must not be activated.

B_1 : **____++++____++++**

Figure 3.9: Target activation pattern. As in Bentley's work [Ben04b], the fitness function had to be modified for FGRN genomes to be reliably evolved to produce this activation pattern. This modification consisted in subtracting to the fitness value the absolute difference between the number of activated-deactivated switches of the pattern (three in this pattern), and the number of those switches in the pattern produced by the tested FGRN; this additional hint favoured FGRNs producing the right "shape" of pattern.

Free floating proteins

The proteins produced by the regulatory genes of a FGRN have often been presented as free floating in the equivalent of a cytoplasm [Ben09] [Ben04a]; and much emphasis is put on the structural similarity of FGRNs with a biological cell. However that is not an accurate description of the workings of the system. In the FGRN system, a separate protein concentration is maintained not for each regulatory protein, but for each regulatory gene. This difference is significant, as regulatory genes producing identical output regulatory proteins are not in practice not rare. There are two reasons for this:

- At the beginning of the search, all the proteins of the randomly initialised genomes of the initial population are set to one of ten pre-evolved proteins and the gene duplication mutation operator.
- The gene duplication mutation operator can create duplicates of regulatory genes with the same output protein. Such gene duplication is common in biology and is an important force in genotypic evolution [TB04]. The ability of natural GRNs to perform consistently after a gene duplication event allows for the evolution of complex behaviours through duplication/divergence mechanisms.

Attempting to follow strictly Bentley's published descriptions of the FGRN system, a version of the system was implemented with free-floating proteins; the concentrations of identical proteins produced by different regulatory genes were then merged together: the output of these regulatory genes were both added to the current protein concentration, and decay was applied to the resulting total protein concentration.

Experiments were run on both the short activation patterns described in Figure 3.8, and on the π problems described in the next sections. The use of free-floating proteins led to a significant and important decrease in the system's performances, both in terms of the maximum fitness reached, and the number of generations it took to reach that fitness. This inability of the FGRN to perform well with free-floating proteins raises doubts on Bentley's claims of structural similarity of the FGRN with biological cells.

3.3 Experiments

Before moving to control problems, the capability of the FGRN system to produce an arbitrary, irregular, phenotype without the help of external inputs will be tested. A good performance here would make it likely that the FGRN system will be able to generate complex control action patterns when required.

Additionally, different output-to-phenotype mapping strategies will be studied. The output of an FGRN system can be used in different ways to produce the final phenotype; in particular, the output of the FGRN system at each running iteration can affect a variable number of components of the phenotype. Two output-to-phenotype mapping strategies opposed on this scale are studied (both mappings are detailed in the Experimental Setup section below):

- **Binary representation.** With this mapping, each bit of data of the phenotype is produced by the output of just one developmental iteration,
- **Approximation algorithm.** With this mapping, the outputs of each developmental iteration influence multiple components of the phenotype throughout development.

The results show that a more dilute approach, letting the output of the FGRN system influence the phenotype in multiple ways, can be advantageous. It is foreseen this could be of use when applying the system to hard control problems.

3.3.1 FGRN evolvability: π as a phenotype

π is an irrational number: its digital representation contains no infinitely repeating pattern. It is also highly irregular; as a phenotype, π is complex, but non-random. Algorithms exist for its production, some are sums of diminishing terms giving an increasingly accurate estimation of π (e.g. $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ...$). Expressed in binary representation, the first 32 bits of π are : 1100 1001 0000 1111 1101 1010 1010 0010.

Developmental systems have often been used to produce repetitive, regular phenotypes; but have not been proven to be as useful for irregular phenotypes. It is therefore also an interesting challenge to try to produce an extremely irregular phenotype with a developmental system.

FGRN genomes will be used as developmental systems, and evolved to produce π. An obvious approach will first be attempted, directly using the FGRN output, each running iteration producing one bit of data of the phenotype π. An indirect approach will then be attempted, giving the FGRN system more freedom, by aiming to produce instead an algorithm approximating the value π. Ideally, it is hoped to obtain an FGRN genome able to approximate π with increasing precision, even when run beyond the number of iterations on which it was evolved.

3.3.2 Experimental setup

Experiment 1 - binary representation

FGRN genomes will be evolved to produce the binary representation of π as a temporal activation pattern. The fitness of the genomes is the number of bits correctly produced by the system before the first error.

Table 3.3: Median and fittest π approximations obtained for each experiment

	Median	**Fittest**
Experiment 1	3.1411	3.1415926
Experiment 2	3.141592658	3.141592658973593

FGA, the genetic algorithm described in Section 3.1.2 is used. The GA settings are the same as in previous FGRN work. Copy and delete mutations are used on genes at the level of the genome, and creep mutation (adding or subtracting a small random value) is applied to the parameters. All mutations occur with a 0.01 probability, which was found in preliminary experiments to give produce fitter individuals than the previously used 0.02 value; crossover is always applied. The population contains a hundred individuals, including a child population of eighty. The initial population will contain randomly generated FGRN genomes with one environmental gene, one receptor gene, four regulatory genes, and one behavioural gene. If a mutation removes the behavioural gene the output of the resulting genome is always zero. The GA is run for a thousand generations and a hundred runs are executed.

Experiment 2 - approximation algorithm

FGRN genomes are evolved as algorithms approximating π. At each developmental iteration, the output of the first behavioural gene will be used as a scaling factor; the mean of the output of the other behavioural genes will be divided by the product of this and the previous scaling factors. If a scaling factor is equal to zero, it will be ignored. If a genome contains zero or one behavioural gene, the total output will be zero. The final approximation is the sum of these terms, over all iterations. This process of generating an approximation p of π is formalised in the following equation:

$$p = \sum_{i=1}^{T} \frac{\frac{\sum_{n=2}^{N} b_{n,i}}{N-1}}{\prod_{t=1}^{i} b_{1,i}}$$

where:

- T is the total number of developmental iterations.
- N is the number of behavioural genes.
- $b_{n,i}$ is the output of the n^{th} behavioural gene at iteration i.

The FGRN genomes are evaluated on thirty-two iterations. The fitness of a solution is the absolute difference between the final approximation p and π. All other experimental settings are identical as in Experiment 1.

3.3.3 Results

Results 1 - binary representation

The fittest FGRN genome evolved produced the binary representation of π to up to twenty-six bits (see Figure 3.10 and Table 3.3).

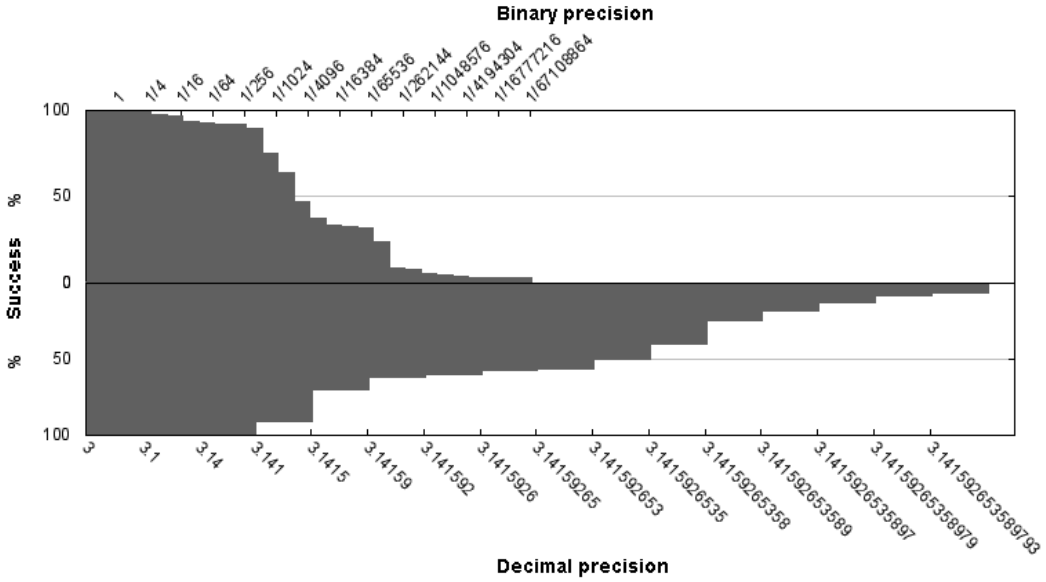


Figure 3.10: FGRN for π : For each degree of precision, the percentage of runs that have reached it. Top: Results for Experiment 1. Bottom: Results for Experiment 2. The binary scale above corresponds to the decimal scale below in terms of accuracy.

Results 2 - approximation algorithm

The fittest FGRN genomes evolved reached the limit of the precision of the datatype initially used, which was floating point double on sixty-four bits. The median number of decimal places reached was eight (thirty-three bits) : 3.141592658. The highest number of decimal places reached was fifteen (fifty-two bits) : 3.141592653589793. More detailed results are displayed in Figure 3.10 and Table 3.3.

The experiment was rerun, using a datatype of unlimited precision for computation of the approximation p; the fittest FGRN genome reached twenty-three decimal places. However, the output of the algorithm diverged from π when run for more than the thirty-two iterations on which it was evolved. It is therefore likely the specific properties of π where not exploited to generate the approximations, but that the overall system was flexible enough instead to produce an accurate approximation of an "arbitrary" number.

Evolving FGRN genomes on a variable number of developmental iterations at each fitness evaluation was attempted to obtain an algorithm that would keep on producing better approximations for as long as it is run. This was unsuccessful, the resulting evolved FGRN genomes producing poorer approximations. In some extreme cases, the evolutionary process got stuck in a local minima producing the exact value of zero, by removing one of the behavioural genes.

Example result

The output of an FGRN genome from Experiment 2, producing π to the maximum precision of the floating point double datatype is shown Figure 3.11 (approximation p) and Figure 3.12 (behavioural genes output). Repeating patterns can be seen in both figures.

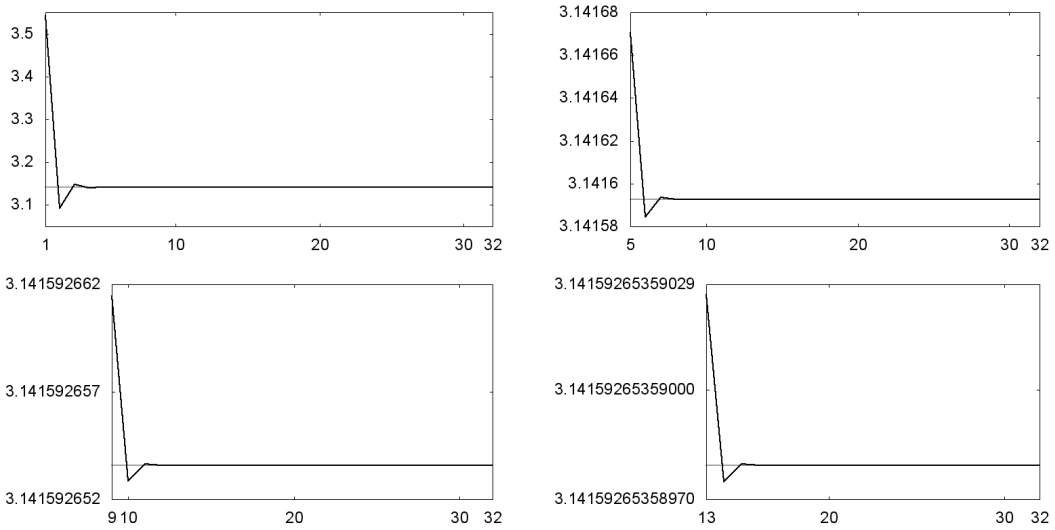


Figure 3.11: The approximation of π throughout the 32 development iterations, from an FGRN genome producing an approximation exact to the maximum precision allowed by the double datatype. The same pattern is repeated at increasingly smaller scales, as development occurs. On each graph the horizontal axis is the number of developmental iterations, and the vertical axis is the associated approximation value.

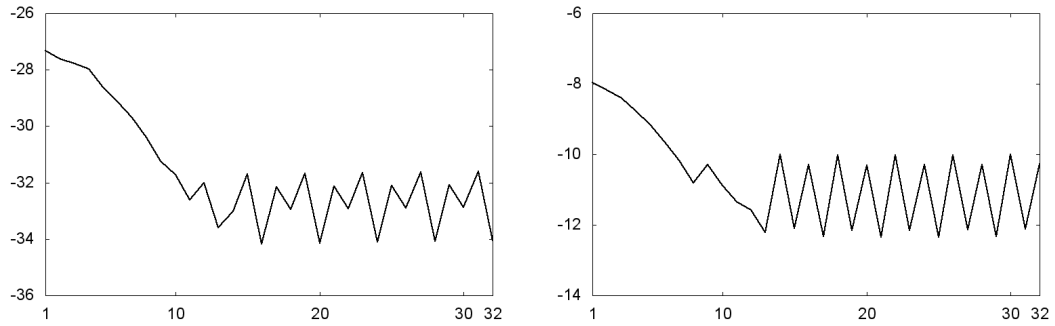


Figure 3.12: Left: the output of the first behavioural gene, the inverse of which is used as a scaling factor in the approximation p. Right: the mean of the outputs of the other behavioural genes, which is the main component of p . The horizontal axes show the number of developmental iteration, and the vertical axes the associated value.

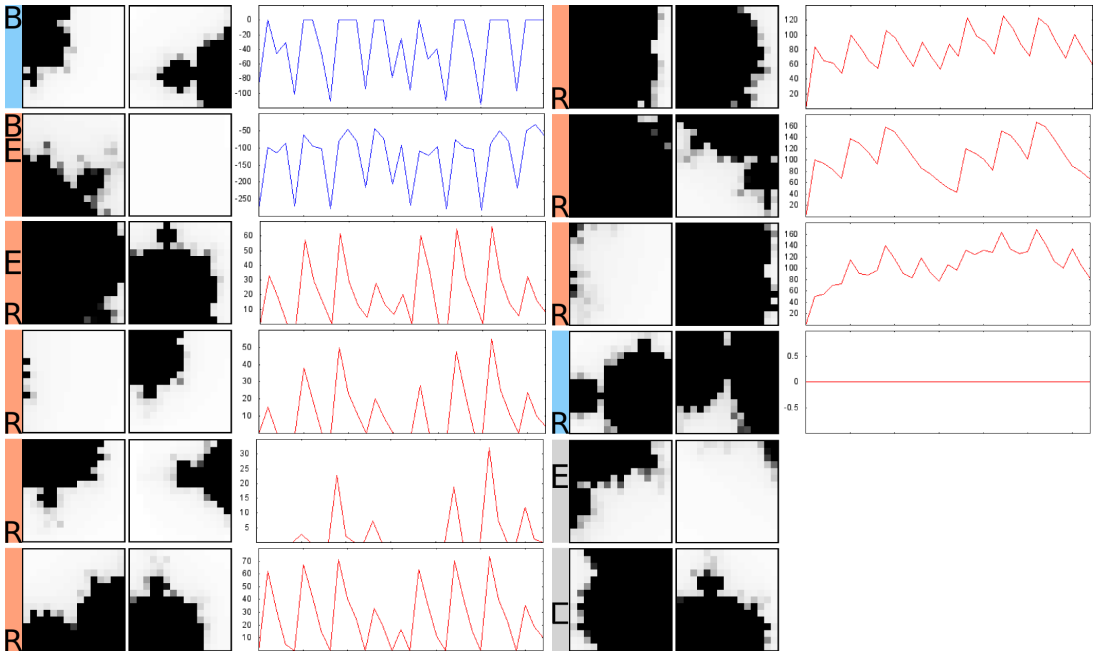


Figure 3.13: The individual genes and their promoter and output proteins, with concentrations of their output proteins over time (B = behavioural, C = receptor, R = regulatory, E = environmental). A blue bar on the left of a gene indicates a positive affinity threshold, and a red bar a negative affinity threshold; genes with a grey bar on the left have an affinity threshold, but it is not used by the running system (e.g. environmental, receptor genes).

Running system

Figures 3.13, 3.14 and 3.15 show the FGRN genome running as a developmental system, illustrating another solution that approximated π to fifteen decimal places in only nine iterations.

3.4 Discussion

FGRN genomes were evolved to produce approximations of π with two different developmental approaches. It was shown that an indirect approach, giving the FGRN system more freedom in the way it influences the final result, by letting it exploit its internal patterns, was more effective for this than the more obvious direct approach.

The attempt to obtain an algorithm that could indefinitely produce increasingly better approximations of π was unsuccessful, but the precision reached by the fittest solutions was impressive, exceeding the precision of the double floating point datatype initially used. This constitutes a further demonstration of the evolvability of the FGRN system from Bentley's FGRN developmental work [Ben04b] [Ben05].

Figure 3.14: The merging of the proteins present in the cell at each developmental iteration, with their associated concentration bitmap. For each iteration, the merging is shown above, and the concentration bitmap below. Iteration 0 corresponds to the state of the cell before the first iteration, at which point it is a function of only the environmental and receptor genes. Several overlapping patterns can be observed in both the merging and the concentration bitmap.

Cycle	Output	Prg.	Pos.
1	3.141309737423218		
2	3.141309737423218		
3	3.141599555932881		
4	3.141592437246917		
5	3.141592653590094		
6	3.141592653590094		
7	3.141592653590094		
8	3.141592653589786		
9	3.141592653589793		
10	3.141592653589793		
11	3.141592653589793		
12	3.141592653589793		
13	3.141592653589793		
14	3.141592653589793		
15	3.141592653589793		
16	3.141592653589793		
17	3.141592653589793		
18	3.141592653589793		
19	3.141592653589793		
20	3.141592653589793		
21	3.141592653589793		
22	3.141592653589793		
23	3.141592653589793		
24	3.141592653589793		
25	3.141592653589793		
26	3.141592653589793		
27	3.141592653589793		
28	3.141592653589793		
29	3.141592653589793		
30	3.141592653589793		
31	3.141592653589793		
32	3.141592653589793		

Figure 3.15: Example result: the approximations produced by the FGRN at each developmental iteration. ‘Cycle’ is the current iteration; the background colour displays changes in the approximation value (light blue: increased, light red: decreased, white: no change). ‘Output’ is the current approximation. ‘Prg.’ indicates the progression of the approximation (blue: better, red: worse). ‘Pos.’ indicates the position of the approximation with respect to π (blue: above, red: under). Note that the precision of the internal floating point representation is exceeded by developmental iteration nine, so the system is unable to improve further.

Chapter 4

FGRNs for Control

Following the presentation of the FGRN model in the previous chapter, and the demonstration of its application to the developmental problem of generating π, the adaptations of the FGRN model allowing it to exploit external inputs for use in control will be detailed in this chapter, before the model is then applied to multiple versions of the pole balancing problem. The pole balancing problem (described in detail in Section 2.1.2), is a well known benchmark control problem, that of balancing a free-swinging pole on a moving cart (also known as the inverted pendulum problem) on a finite track by pushing the cart left or right.

Modifications to the model and in the genetic algorithm employed to evolve the genomes will be shown to improve performance, in both the speed and reliability with which a successful controller is found. After each change the modified system is then re-evaluated on the pole balancing problems. In summary the following modifications are made to improve the system's performance:

- An alternative genetic algorithm to the one used in all other FGRN work is used, which improves the reliability with which a successful controller is found.
- Aiming to discard unneeded physical constraints, negative concentrations are introduced to represent negative inputs, a more straightforward representation.
- A change in the FGRN algorithm, which was present in Bentley's initial work [Ben04b], but was disabled in Bentley's further work [Ben05], will be implemented. It consists of adding a supplementary condition to the activation of behavioural genes, based on the concentration of the cytoplasm "protein". The difference in the system's behaviour, and the significant resulting improvement in performance, will be discussed.

The performance of the system will then be compared with that of other control systems; specifically, simple recurrent neural network (RNN) controllers are evolved and tested in the exact same conditions, for comparison.

The work detailed in this chapter was at the time of publication one of the first applications of a GRN model to a substantial, well-recognised, control problem, closely following after Nicolau et al.'s work [NSB10], which used a different problem setup using randomised cart-pole starting positions, and focused more on ensuring the successful solutions found worked for as much of the state-space of

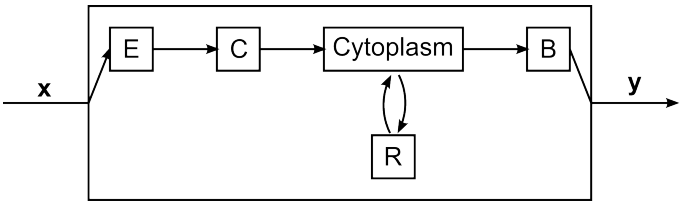


Figure 4.1: The workings and role of each gene type in an FGRN controller. It is very similar to Figure 3.3 in the last chapter. '**x**' represents the inputs to the controller, and '**y**' the outputs. Key: **E**: environmental, **C**: receptor, **R**: regulatory, **B**: behavioural.

starting positions as possible. In terms of the number of failures before successful control, which is equivalent for evolutionary systems to the number of evaluations until a good solution is found and is the most commonly used performance metric for this problem, the performances obtained by Nicolau et al.'s system were also not as good as those obtained here. Some simple, non-standard control applications had been attempted by Bentley [Ben03a] and Zahadat et al. [ZK08]. However an application of the system on a standard, difficult, control problem had yet to be attempted.

4.1 Initial system adaptations for control

The FGRN model is essentially the same when used for control as when used for development, the main difference residing in the acceptance of external inputs. In keeping with the biological metaphor, the concentrations of environmental proteins are mapped to the input values. Similarly to developmental use, the genes of a control FGRN genome can have any combination of the same four types. However some gene types now have different roles due to the necessity of integrating external inputs into the system's workings:

- **Environmental** genes provide input to the system. There is one environmental gene per component of the input vector $\mathbf{x}$. Each input component here is mapped to the $\mathbb{R}$ range $[0, 1]$, as it represents a concentration linked to its environmental gene. If, for instance through mutation, the number of environmental genes is lesser than the number of inputs, the first inputs are associated with the existing environmental genes, and the remaining inputs are ignored by the system.

- **Receptor** genes act as an input filter. A receptor gene allows for part of each of the environmental proteins, acting as inputs, to be merged into the controller's internal state. This allows the system to reserve part of the internal state of the controller for internal computations only.

- **Regulatory** genes perform internal computations based only on the internal state (cytoplasm) of the system, which however is itself influenced by the latest inputs.

- **Behavioural** genes each produce a controller output, based on the current internal state of the system.

At any given timestep, information from the environmental inputs is processed simultaneously by the regulatory and behavioural genes. Behavioural output is therefore a function of current environmental

inputs and of the output of regulatory genes at previous timesteps.

Information from the inputs is only processed on the same timestep by behavioural genes; the output of the regulatory genes only affecting the output of the controller from the next time step onwards. The role of each of the gene types is further illustrated in Figure 4.1. The internal structure of a sample FGRN controller is illustrated in Figure 4.2. The FGRN main loop adapted for control is detailed in Algorithm 4.1 (see Algorithm 3.2 for the developmental equivalent; the same functions, detailed in Section 3.1.5 in the previous chapter, are used in both pseudocode listings). Note the use of the behavioural gene activation state as boolean output, as opposed to the real output value as used in development, as a binary output is more suitable to the control problems considered, which all use "bang-bang" control. Algorithm 4.1 contains the following additional functions relating to interactions with the controlled system:

- getInputs(): get controller inputs from the controlled system.
- setOutputs(outputs): set the controller outputs, which direct the controlled system.
- controlFailure(): return true until the controlled system fails or terminates, then false.

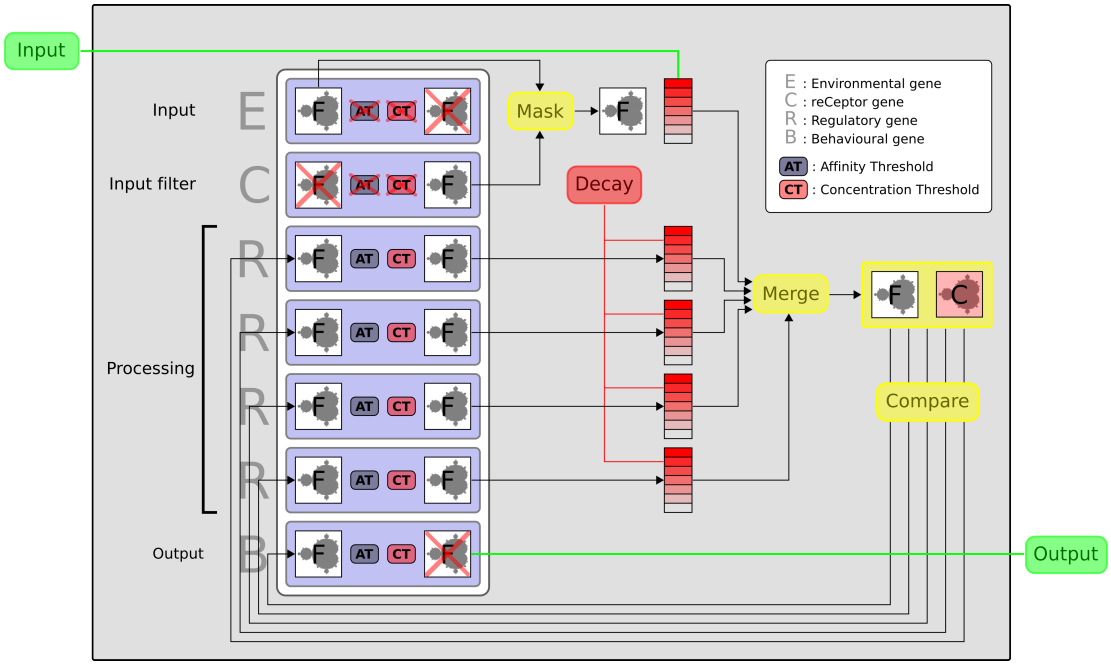


Figure 4.2: Internal view of an example FGRN controller with one input and one output. The genome, a series of genes (in blue) is shown on the left. A red cross over an element of a gene means that it is not used in the running of the system. The white to red gradients are protein concentration scalars, and vary during the running of the system. The protein chemistry operations mask, merge and compare are shown in yellow, as well as the merging of proteins (cytoplasm), shown on the right. Decay, another chemistry operation, is shown in red, as it only affects the concentration scalars. For an "input" protein, the shape of the protein is determined from the fractal shape of an environmental protein (masked by the receptor promoter protein) but its associated concentration is determined from the external input value.

Algorithm 4.1: The FGRN control main loop. Key: AT = affinity threshold, CT = concentration threshold.

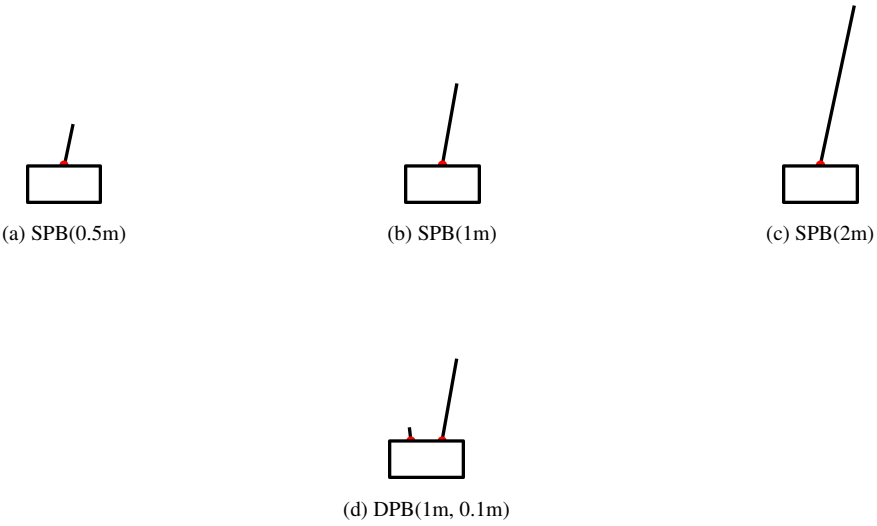
```
{ split the genome into arrays of genes }
declare array environmental_genes := getEnvironmentalGenes(genome)
declare array behavioural_genes := getBehaviouralGenes(genome)
declare array regulatory_genes := getRegulatoryGenes(genome)
declare struct receptor_gene := getReceptorGenes(genome)[1]
declare struct input_mask := receptor_gene.output_protein

{ one iteration covers one input−output cycle }
while not controlFailure()
  declare array proteins_to_merge := []

  { integrate inputs as environmental concentrations }
  declare array inputs = getInputs()
  for i := 1 in inputs.length
    if inputs[i] > 0
      append(proteins_to_merge, <mask(environmental_genes[i].promoter_protein, input_mask), inputs[i]>)
    end
  end

  { decay regulatory gene concentrations, merge into cytoplasm if still present }
  for each gene in regulatory_genes
    decayConcentration(gene)
    if gene.concentration > 0
      append(proteins_to_merge, <gene.output_protein, gene.concentration>)
    end
  end

  declare struct cytoplasm := merge(proteins_to_merge)

  { activate regulatory and behavioural genes }
  for each gene in regulatory_genes, behavioural_genes
    gene.activated := false
    <ΔP, c̄> := compareCytoplasmToPromoter(cytoplasm, gene.promoter_protein)
    p_a := activationProbability(gene.AT, ΔP)
    if rand() < p_a          { stochastic activation }
      gene.activated := true
      if gene.is_regulatory
        gene.concentration += regulatoryConcentrationUpdate(gene.CT, c̄)
      end
    end
  end

  declare array outputs := [ ]
  for each gene in behavioural_genes      { extract bang−bang output values }
    append(outputs, gene.activated)
  end
  setOutputs(outputs)
end
```

4.2 Experiments: pole balancing

As discussed in the literature review, the pole balancing problem is a widely used benchmark control problem. The details of the pole balancing problem, including equations of motion and the acceptable parameter range for successful control, are given in the literature review, Section 2.1.2.

The FGRN model's control ability will be tested on a range of variants of the pole balancing problem. The mechanical variations are detailed in Figure 4.3. Control will be attempted with both full-state input, in which both position and velocity of the cart and pole are provided to the controller; and with partial inputs, the controller being then only provided with the position of the cart and pole(s), and not their velocities. The activation state of a single behavioural gene will be mapped to the controller output which belongs to the set $\{-1.0, 1.0\}$. The fitness of a controller is the number of timesteps for which it keeps the pole(s) balanced while keeping the cart within the track's limits.

The controller inputs with the associated scaling factors used for single pole balancing are shown in Tables 4.1 and 4.2 for full-state and partial inputs, respectively. Similarly for double pole balancing the inputs used are shown in Tables 4.3 and 4.4. These scaling factors bring each input into the $[-1, 1]$ range, the result being then linearly mapped to the $[0, 1]$ concentration range, which is the current range of input for the FGRN system. The scaling factors were obtained from the maximal possible values for the positional inputs, and from the maximal values experimentally observed for the velocity inputs.



(a) SPB(0.5m) (b) SPB(1m) (c) SPB(2m)

(d) DPB(1m, 0.1m)

Figure 4.3: Variations of the pole balancing system studied. a) Single pole balancing with half metre pole: SPB(0.5m). b) Classical single pole balancing one metre pole: SPB(1m). c) Single pole balancing with two metre pole: SPB(2m). d) Classical double pole balancing with one metre and 10 centimetre poles: DPB.

Table 4.1: The controller inputs for single pole balancing, with full-state inputs (SPB).

Position	Value	Scaling
1	x	$\frac{1}{2.4}$
2	$\dot{x}$	$\frac{1}{6.0}$
3	θ	$\frac{1}{\frac{12\pi}{180}}$
4	$\dot{\theta}$	$\frac{1}{6.0}$

Table 4.2: The controller inputs for single pole balancing, with positions only, no velocity inputs (SPB(NV)).

Position	Value	Scaling
1	x	$\frac{1}{2.4}$
2	θ	$\frac{1}{\frac{12\pi}{180}}$

Table 4.3: The controller inputs for double pole balancing, with full-state inputs (DPB).

Position	Value	Scaling
1	x	$\frac{1}{2.4}$
2	$\dot{x}$	$\frac{1}{6.0}$
3	θ_1	$\frac{1}{\frac{36\pi}{180}}$
4	$\dot{\theta_1}$	$\frac{1}{6.0}$
5	θ_2	$\frac{1}{\frac{36\pi}{180}}$
6	$\dot{\theta_2}$	$\frac{1}{24.0}$

Table 4.4: The controller inputs for double pole balancing, with positions only, no velocity inputs (DPB(NV)).

Position	Value	Scaling
1	x	$\frac{1}{2.4}$
2	θ_1	$\frac{1}{\frac{36\pi}{180}}$
3	θ_2	$\frac{1}{\frac{36\pi}{180}}$

4.2.1 Experimental settings

Runge-Kutta fourth order integration, with the usual 0.2s control timestep, and 0.1s simulation timestep are used, as in Gomez et al.'s work [GSM08]. a maximum of 10,000 fitness evaluations are allowed per run, and 50 runs are executed. A controller must balance the pole for 100,000 timesteps ($\approx$ 30 minutes) to be considered successful. The fitness of a genome is simply the number of timesteps for which its associated controller balanced the pole before failure. The usual FGRN GA is used with the same settings as in Chapter 3. The mutation rate is set to 0.1.

4.2.2 Results

Table 4.5 details the results of the pole balancing experiments with the FGRN genomes evolved by the FGA. Successful controllers were found in most cases for the full state pole balancing, and in some cases when given only partial inputs (no velocities). However no successful controller was found for the double pole problem, even when the full inputs were given to the controllers.

The figures in the following four pages display the performance of the system in each one of fifty runs, on each of the problems defined above. Each blue line represents the performance improvements in fitness in one run of the GA, as a function of the number of fitness evaluations so far. The thick red line represents the median fitness over all runs. The equivalent in terms of GA generations is the number of evaluations divided by the number of fitness evaluations per generation (a hundred).

Table 4.5: Results for the FGRN model on the pole balancing problems, with FGA search. The first, '%' column gives the percentage out of the fifty runs which produced a successful controller. The next two main columns, each subdivided into median, mean and standard deviation(SD), give detail of the final fitness at the end of the runs, and, in case a successful controller is reached, the number of evaluations it took to reach it. Key: SPB = single pole balancing, DPB = double pole balancing, NV = no velocities.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	94%	100000	94103	23343	2612	3296	2197
SPB(1.0m)	82%	100000	82513	37342	2140	2673	1614
SPB(2.0m)	86%	100000	87147	32195	2206	2607	1392
DPB	0%	80	79	35	-	-	0
SPB(0.5m) NV	24%	251	24221	42584	5982	6038	2324
SPB(1.0m) NV	20%	250	21918	39619	5112	4993	2284
SPB(2.0m) NV	22%	253	22229	41303	3720	5109	2407
DPB NV	0%	35	38	7	-	-	0

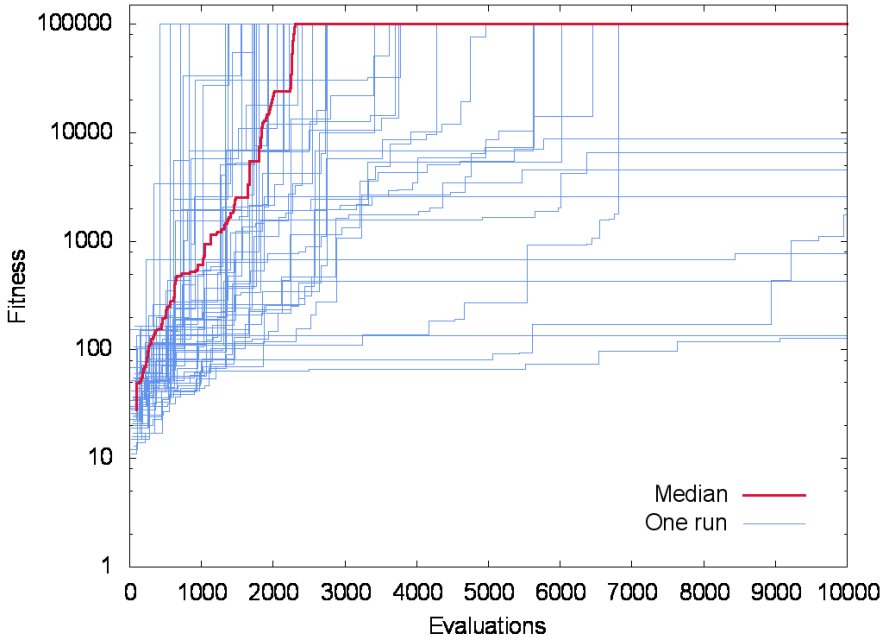


Figure 4.4: FGRN model with FGA learning on the single pole balancing problem with 1.0m pole (SPB(1.0m)). Each blue line represents one of the fifty runs, and the bold red line is the median of these runs at each point.

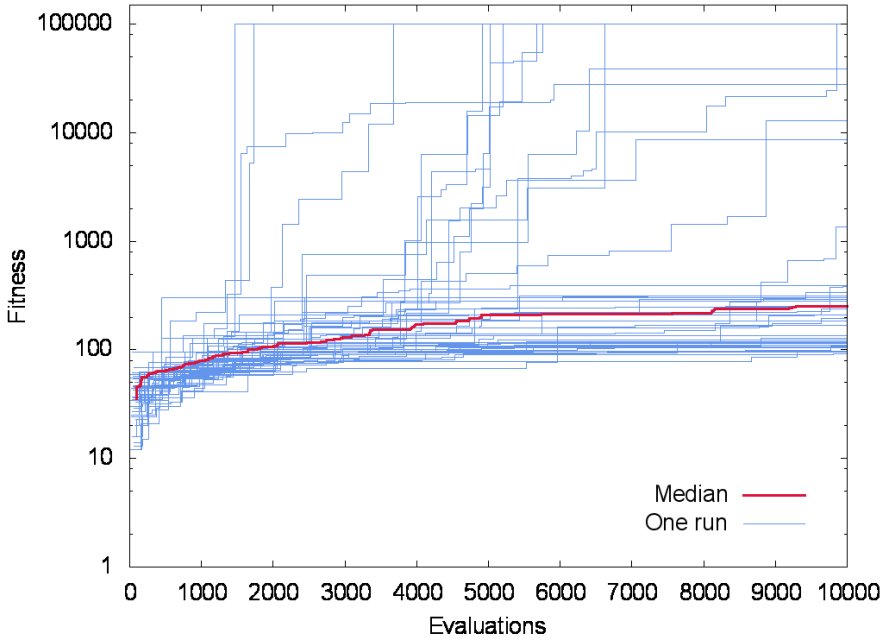


Figure 4.5: FGRN model with FGA on the single pole balancing problem with 1.0m pole and no velocity inputs (SPB(1.0m) NV).

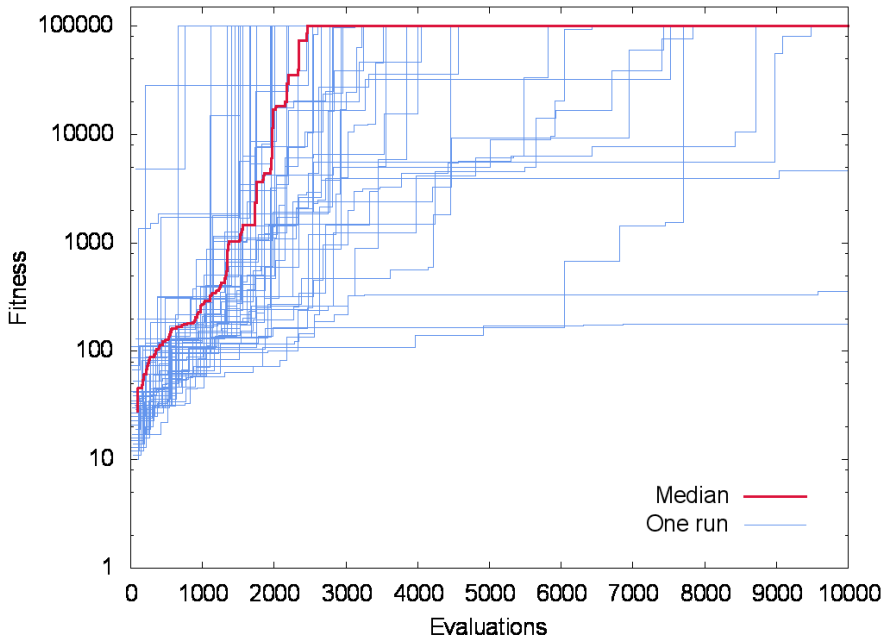


Figure 4.6: FGRN model with FGA on the single pole balancing problem with 0.5m pole (SPB(0.5m)).

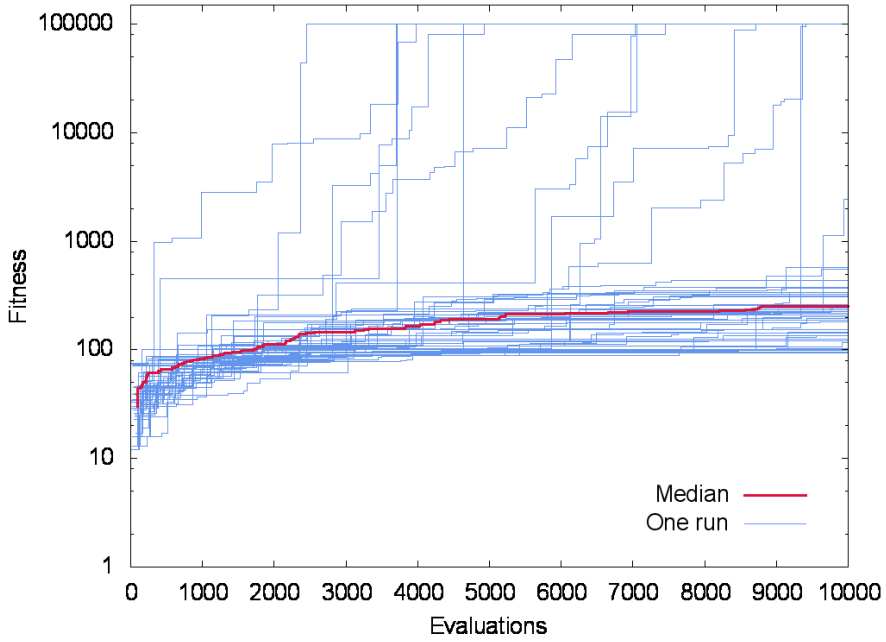


Figure 4.7: FGRN model with FGA on the single pole balancing problem with 0.5m pole and no velocity inputs (SPB(0.5m) NV).

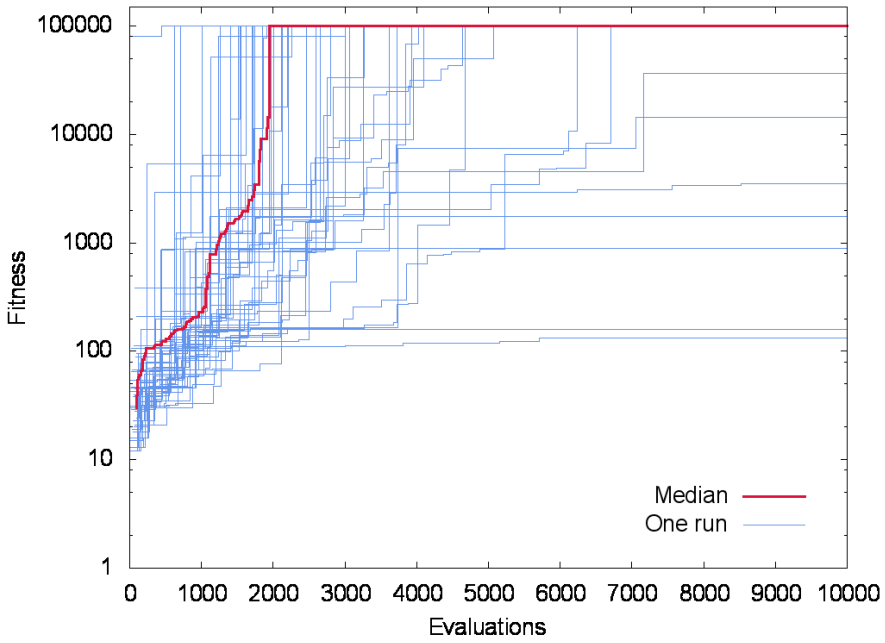


Figure 4.8: FGRN model with FGA on the single pole balancing problem with 2.0m pole (SPB(2.0m)).

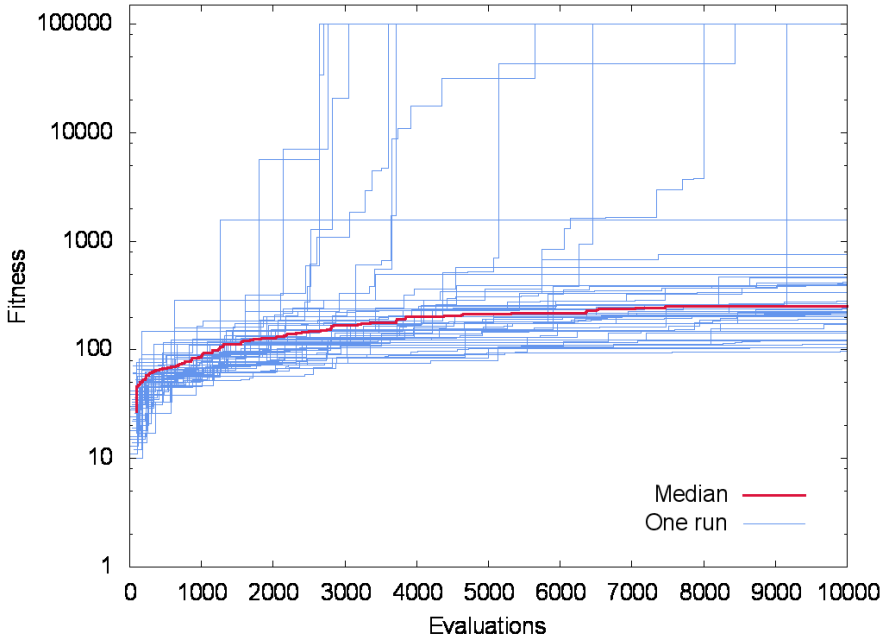


Figure 4.9: FGRN model with FGA on the single pole balancing problem with 2.0m pole and no velocity inputs (SPB(2.0m) NV).

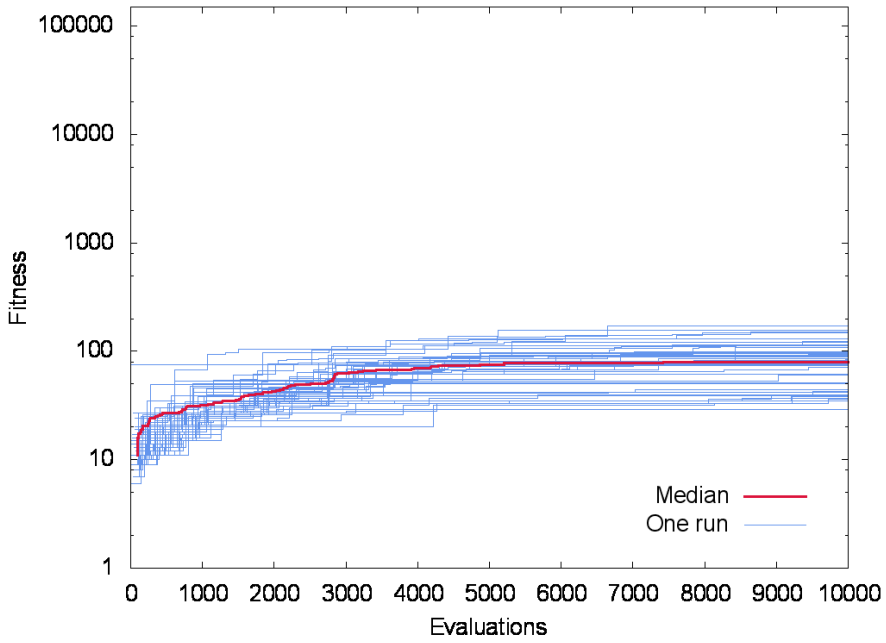


Figure 4.10: FGRN model with FGA on the double pole balancing problem (DPB).

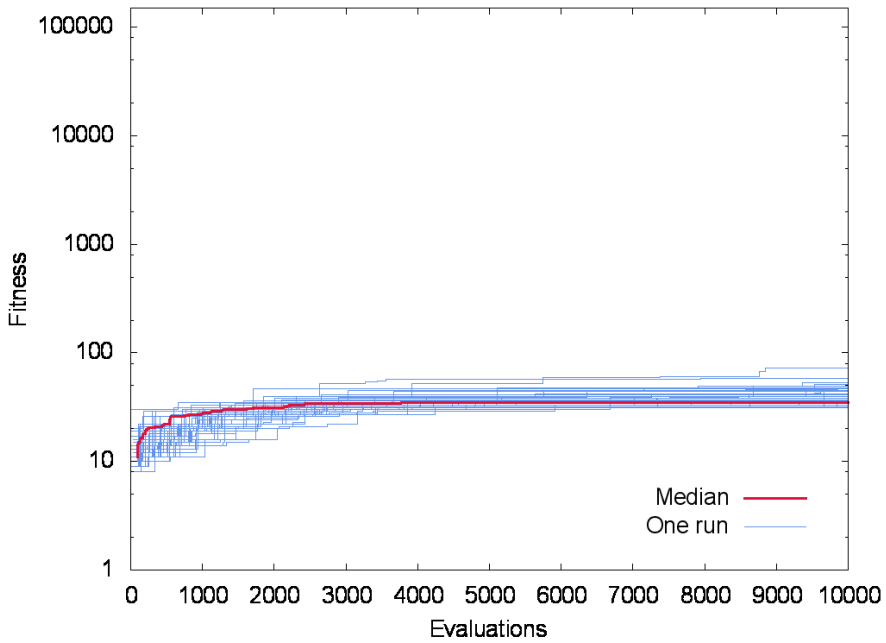


Figure 4.11: FGRN model with FGA on the double pole balancing problem with no velocity inputs (DPB NV).

As can be seen from the graphs, the length of the pole in the single pole problem has little effect on the results, whereas the absence of velocities make the problem considerably harder. The system did not come close to finding a solution to the double pole balancing problems, with or without velocities.

4.3 Improving reliability with ALPS

One issue with the performances detailed above is the reliability with which successful solutions are found across the runs. This is particularly apparent in the single pole balancing with velocity experiments; in some runs a local optimum is reached early on and is improved very little or not at all during the rest of the run. This corresponds to in Figures 4.4, 4.6, and 4.8, to the thin blue lines that are straight for most of the run without reaching the successful fitness.

Hornby's Age-Layered Population Structure (ALPS) paradigm [Hor06] for genetic algorithms was created to avoid premature convergence, and was shown to be successful on a variety of applications [Hor09] [PC07]. In this section, a GA implementing the ALPS paradigm will be described, followed by running the pole balancing experiments above using this GA. The results will then be compared with the previous FGA results; it is hoped the use of ALPS will reduce premature convergence issues.

4.3.1 ALPS description

The ALPS paradigm aims to preserve diversity in the genetic material by keeping separate populations in age-segregated layers. Amongst ageing individuals, those with a high fitness filter up to the upper layers, while those with a lower fitness stay in the lower layers and are eventually replaced by younger individuals. The population of the first (lowest) layer is regularly (every AgeGap generations) replaced by randomly generated individuals, allowing a constant inflow of new genetic material. An individual's age is only incremented in a generation if it was used as a parent in that generation; the age is then incremented by one, regardless of the number of offspring produced. The age of individuals which were not used as parents does not increase.

Each layer has a limit age, fixed by the products of an ageing scheme series and the value of AgeGap. For example, using an exponential (2^n) ageing scheme and an AgeGap value of 10, the limit age of layer n is 10×2^n generations. If the number of layers is constrained, the uppermost layer has no age limit. When an individual's age exceeds the limit of its current layer if the individual is better than the worst individual of the layer above, it replaces it, otherwise it is discarded. Algorithm 4.2 describes the workings of ALPS. The functions referenced in the algorithm are the following:

- newRandomLayer() - generate a new layer with LayerSize randomly generated genomes.
- evaluateLayer(layer_index) - evaluate the fitness of all the individuals in layer layer_index.
- generateOffspringPopulationLayer(layer_index) - replace layer layer_index by offspring generated from parents from this layer and the layer immediately below. The offspring are generated by the inner-layer GA, described below.

Algorithm 4.2: The ALPS algorithm.

```
declare integer generation := 0
declare struct[] layers := []

{ Initialise first layer }
layers[1] := newRandomIndividuals(LayerSize)
evaluateLayer(1)

{ Run for GenerationCount }
for generation := 1 in GenerationCount

  { generate an offspring population for each layer }
  for i := size(layers) downto 1
    if i == 1 && generation \% AgeGap == 0
      { reset bottom layer every AgeGap generations }
      promoteIndividualsToLayerAbove(1)
      layers[1] = newRandomIndividuals(LayerSize)
    else
      generateOffspringPopulationLayer(i)
      evaluateLayer(i)
    end
  end

  { age all individuals }
  for i := 1 in size(layers)
    for j := 1 in LayerSize
    ageIndividual(layers[i][j])
  end

  promoteIndividualsToLayerAbove(1)

  generation++
end

return layers[size(layers)][1]
```

- promoteIndividualsToLayerAbove(layer_index) - move all individuals which are too old for layer layer_index to the layer above, replacing the worst inferior individuals in the layer above, delete old individuals which are too old and for which there is no space above. Additional upper layers are created as needed.

- ageIndividual(individual) - increase the age of individual.

Each layer runs a separate genetic algorithm. The ALPS algorithm allows a large variety of GAs in that role, provided that the parent population is taken from both the current layer and the layer below. Here, as in Hornby's initial ALPS work [Hor06], tournament selection is used. Each one of an offspring's two parents is chosen by tournament of size four: four candidate parents are randomly selected from the population, the fittest one is chosen. An elitism of four is also applied: the four fittest individuals in the current layers are kept unchanged in the offspring population.

4.3.2 Experiments

The settings for the ALPS GA used here are the following: the `AgeGap` is set to ten, the layer size to twenty-five individuals, with a maximum of ten layers. The age limit of each layer is set using a polynomial layer ageing scheme (1, 2, 4, 9, 16, 25, 36, 49, 64). These parameters where chosen to allow the maximal expression of ALPS mechanisms within the number of fitness evaluations allowed in a run. The other experimental settings are identical to those in the pole balancing experiments above.

Table 4.6 shows the results. The use of ALPS is found to improve the reliability with which a successful solution is found with a low significance ($0.5 < p < 0.1$) across the problems. In practice however, the use of ALPS greatly reduced the amount of computation required. This was due to what happens when FGA converges to a high-fitness, but unsuccessful solution; the population then fills up with such solutions which require a large amount of processing to evaluate (due to the higher cost of simulating the pole balancing system for a long time). ALPS's layered approach restricted such solutions to its higher layer. Therefore ALPS may still be considered to have been a successful modification to the system. However there was no improvement on the performances on double pole balancing.

As in the previous section, the plots in the following four pages detail the results of the experiments on each variation of the pole balancing problem.

Table 4.6: Results for the FGRN model on the pole balancing problems, with ALPS GA search. The organisation is the same as that of Table 4.5.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	96%	100000	96430	17542	2339	2947	2215
SPB(1.0m)	96%	100000	96801	15677	2247	2955	2277
SPB(2.0m)	96%	100000	98127	10845	3081	3177	2096
DPB	0%	69	69	19	-	-	0
SPB(0.5m) NV	28%	3245	32123	43136	6991	6145	2531
SPB(1.0m) NV	14%	264	17216	34437	8822	7973	2037
SPB(2.0m) NV	34%	1497	38408	46815	6155	5551	2276
DPB NV	0%	35	35	3	-	-	0

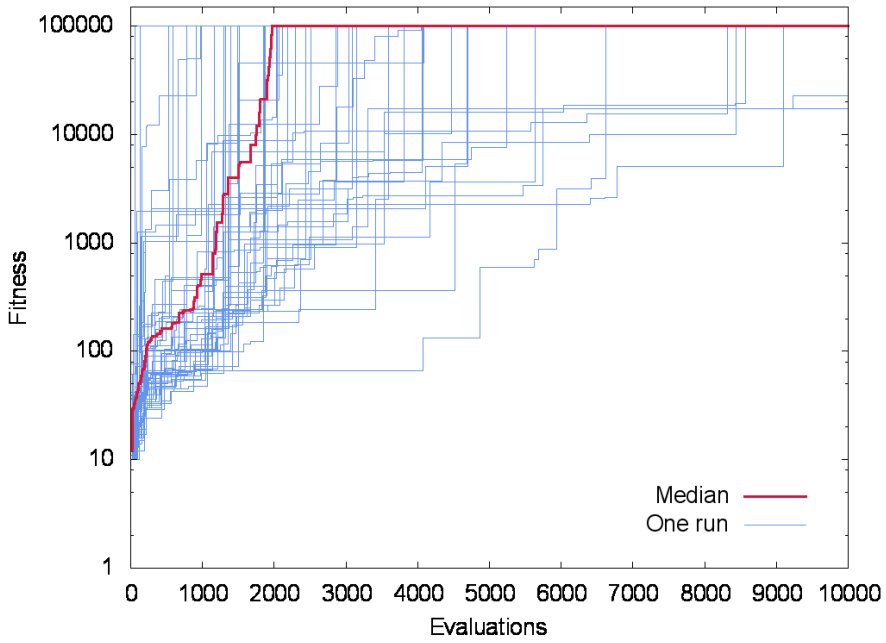


Figure 4.12: FGRN model with ALPS on the single pole balancing problem with 1.0m pole (SPB(1.0m)).

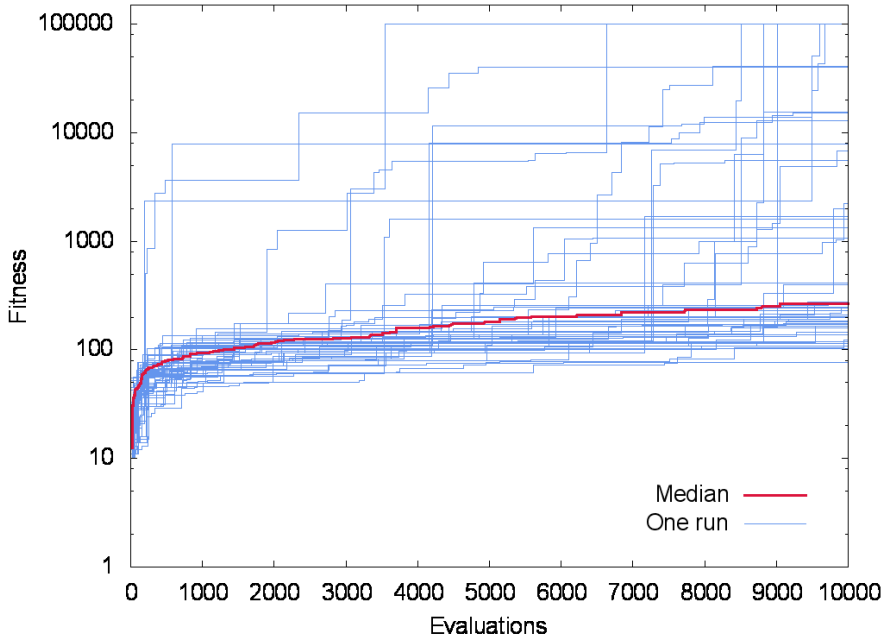


Figure 4.13: FGRN model with ALPS on the single pole balancing problem with 1.0m pole and no velocity inputs (SPB(1.0m) NV).

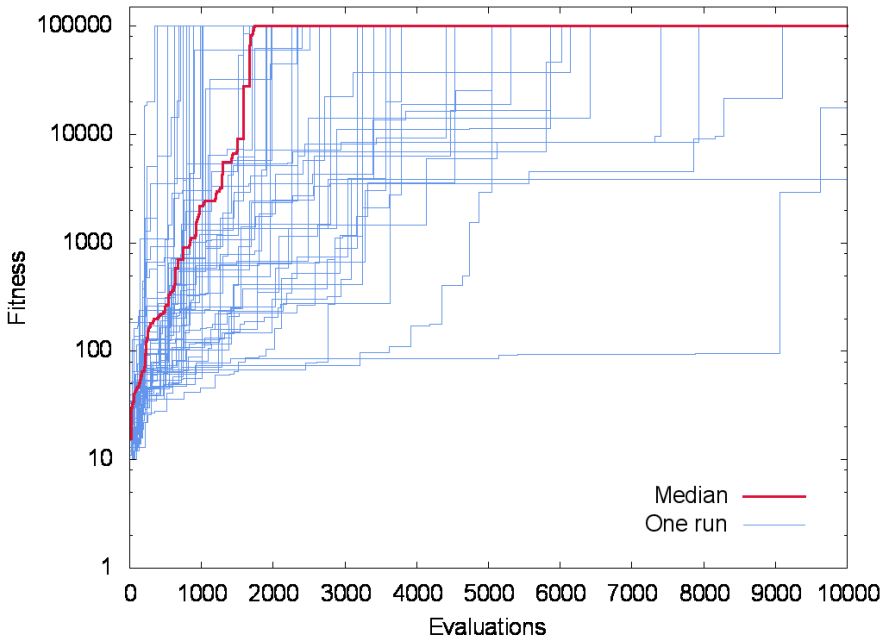


Figure 4.14: FGRN model with ALPS on the single pole balancing problem with 0.5m pole (SPB(0.5m)).

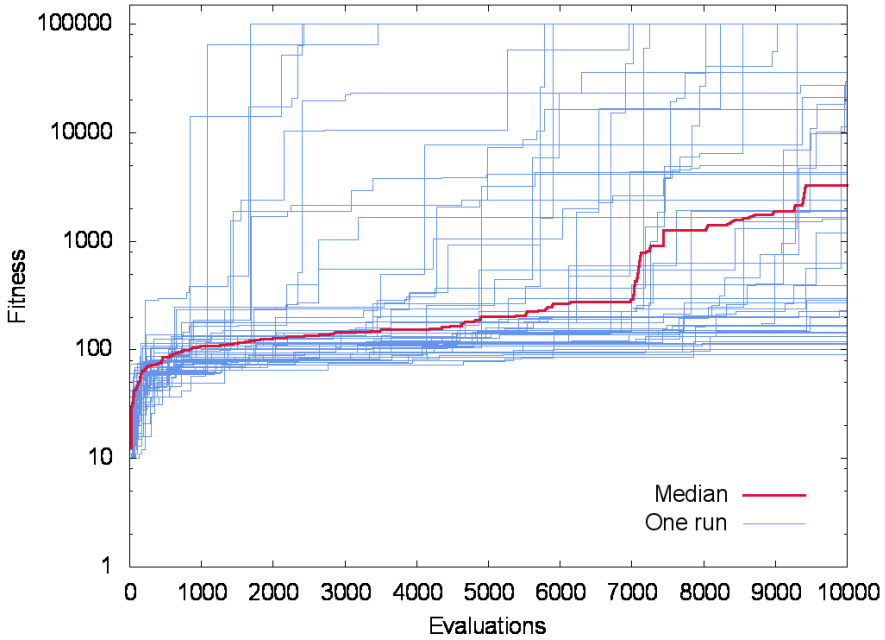


Figure 4.15: FGRN model with ALPS on the single pole balancing problem with 0.5m pole and no velocity inputs (SPB(0.5m) NV).

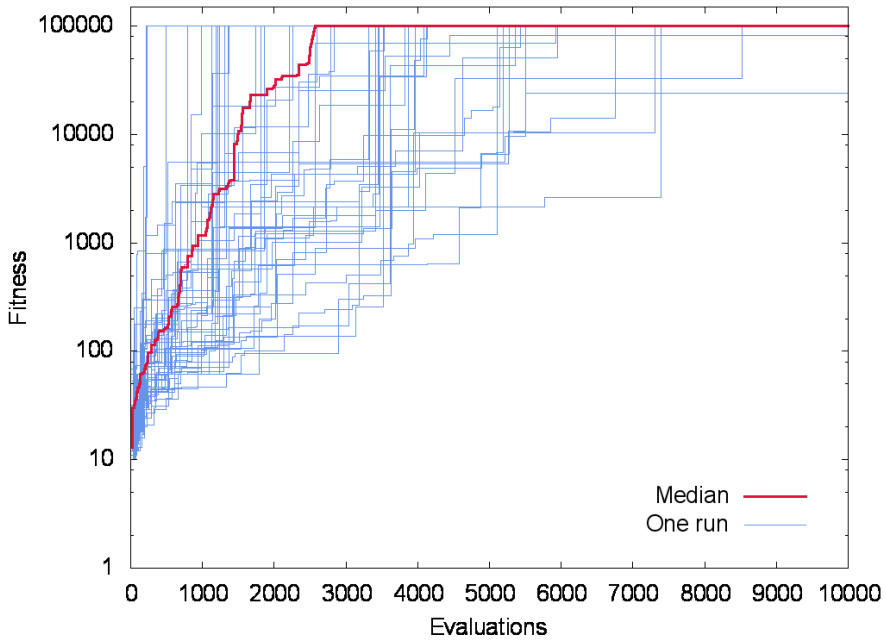


Figure 4.16: FGRN model with ALPS on the single pole balancing problem with 2.0m pole (SPB(2.0m)).

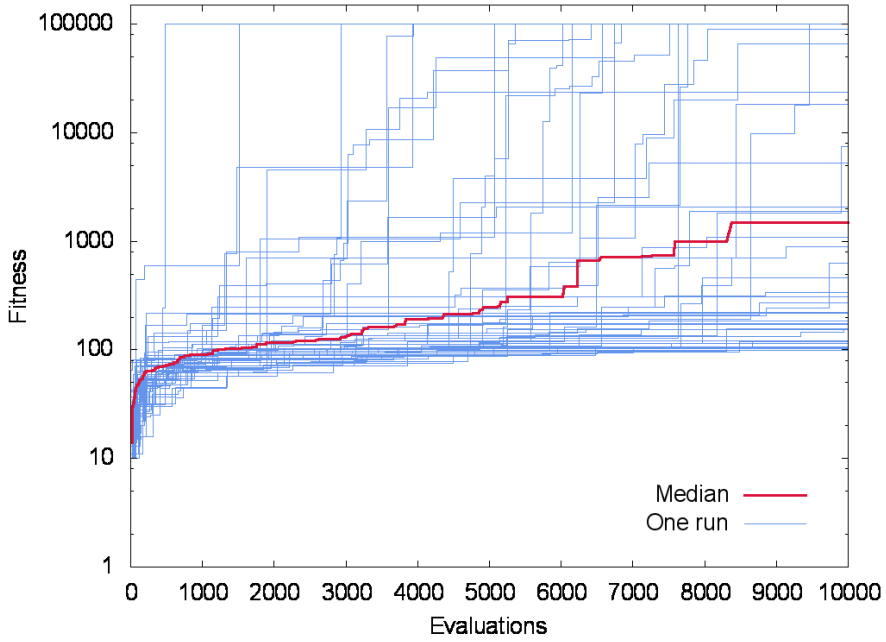


Figure 4.17: FGRN model with ALPS on the single pole balancing problem with 2.0m pole and no velocity inputs (SPB(2.0m) NV).

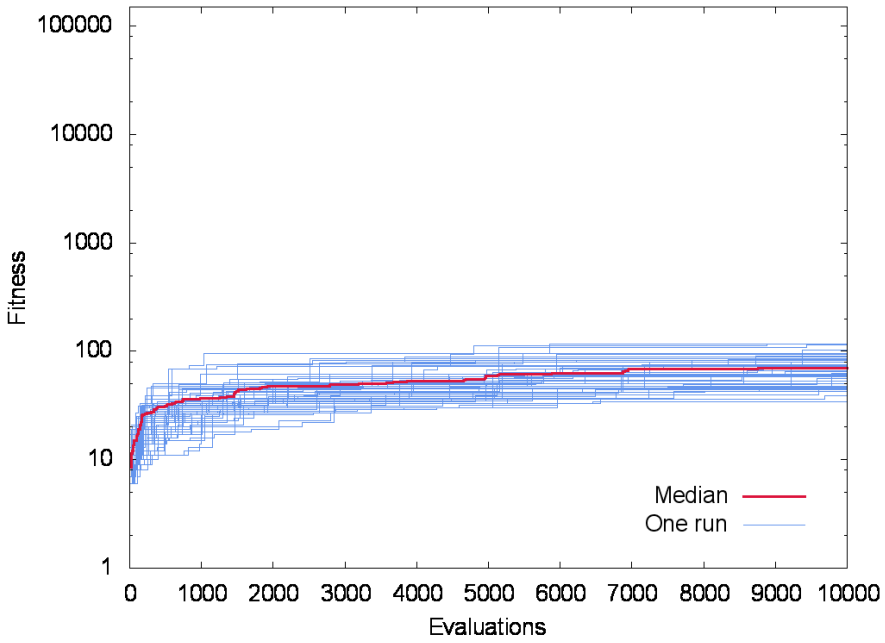


Figure 4.18: FGRN model with ALPS on the double pole balancing problem (DPB).

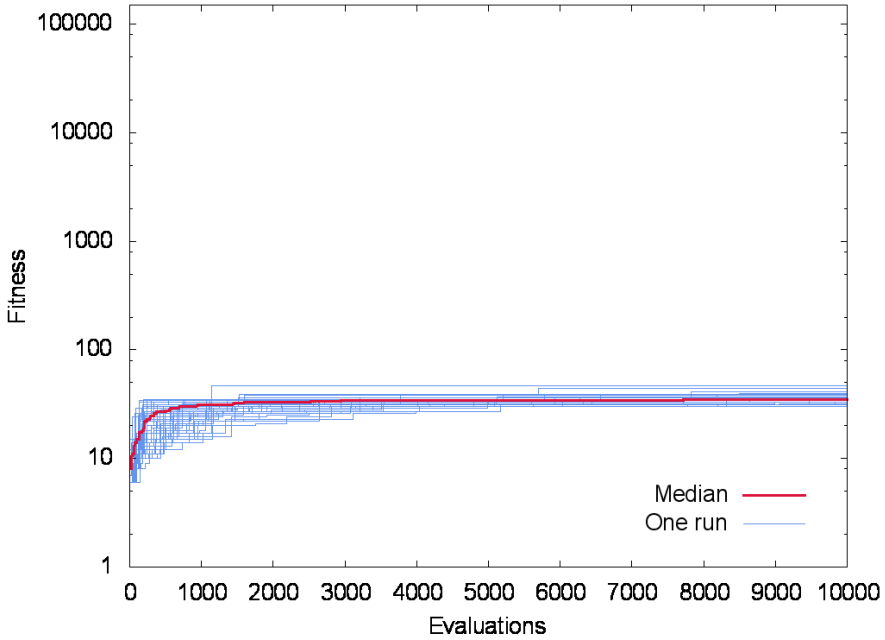


Figure 4.19: FGRN model with ALPS on the double pole balancing problem with no velocity inputs (DPB NV).

4.4 Introducing negative input protein concentrations

Negative concentrations are introduced for encoding inputs : the controller's inputs are here mapped to the range $[-1, 1]$ instead of $[0, 1]$. It is hoped the system will be able to take advantage of the sign of inputs to improve performances. Previous reinforcement learning methods relied on similar partitions of the input state-space to exploit the sign of input values and successfully balance the pole[BSA83]; it would be interesting to see whether the FGRN model can similarly exploit the sign of input values. The concept of negative concentrations is not physically plausible, but their use for inputs does not introduce inconsistencies within the model. This is a similar leap to the use in the perceptron model of negative weights. The new range and accordingly modified settings are detailed in Table 4.7.

Table 4.7: Negative input concentration settings for FGRN control. Following Bentley's work [Ben04b] the normal protein concentration range is $[0, 200]$, 200 representing saturation. For negative concentration this range is extended to $[-200, 200]$. The concentration threshold initialisation range is changed accordingly.

Setting	**Original**	**Negative input concentrations**
Input range	$[0, 1]$	$[-1, 1]$
Concentration threshold initialisation range	$[0, 200]$	$[-200, 200]$

4.4.1 Experiments

The changes pertaining to negative concentrations are applied cumulatively to the previous change to using the ALPS GA. The experimental settings are identical to those used with ALPS above.

Table 4.8: Results for the FGRN model with negative protein concentrations and ALPS GA search on the pole balancing problems. The organisation is the same as that of Table 4.5.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	100%	100000	100000	0	593	756	818
SPB(1.0m)	100%	100000	100000	0	832	957	732
SPB(2.0m)	100%	100000	100000	0	609	676	546
DPB	0%	47	51	12	-	-	0
SPB(0.5m) NV	4%	193	4759	19539	8185	8185	1544
SPB(1.0m) NV	10%	222	10448	29866	4766	5368	2353
SPB(2.0m) NV	14%	195	14335	34574	3283	4310	2597
DPB NV	0%	26	27	3	-	-	0

The change to using negative concentrations greatly improved the performance of the system on full state pole balancing of all length; a successful controller was found on every run and the mean number of evaluations to success on the three pole lengths which previously averaged over 3000 evaluations now

averages under 800 evaluations. However the results on single pole balancing without velocities are still poor, and there is even a small but noticeable decrease in performance. The system still fails completely on double pole balancing. As in the previous sections, the plots in the following four pages detail the results of the experiments on each variation of the pole balancing problem.

Extending input concentrations beyond saturation

From the results above alone, it could be argued that the large increase in performance may not be due to the ability of the system to exploit the negative concentrations, but simply from the increased granularity provided by the wider input range. To resolve this question, an alternative input range is considered: as the input range was extended from $[0, 1]$ to $[-1, 1]$ above, the same set of experiments is run extending the input range $[0, 1]$ to $[0, 2]$. This leads to environmental protein concentrations superior to saturation which, though as impossible in real world terms as negative concentrations, are also tolerated by the FGRN algorithm. However, as for negative concentrations, concentrations above saturation do not propagate throughout the system, i.e. the regulatory concentrations are still kept within the sane $[0, 1]$ range. The results of these experiments are detailed in Table 4.9.

The system with double input range performs significantly worse on the full state pole balancing problems than both the original FGRN version and the version allowing negative inputs. Particularly, for the runs for which it finds solutions, it does so at the expense of several time the number of evaluations than the system with negative input concentrations. However it also presents significant improvements on the partial state pole balancing problems compared to both other versions of the FGRN. Each result comparison above is significant with $p < 0.03$, conservatively calculated by considering for comparison a failed run to be equivalent to a 10,000 evaluations long successful run.

Overall, the input range is found to have a significant influence on the performance of the system. It would be desirable for it to be adaptive, for instance by being subject to evolution; however though possible this doesn't fit well within the FGRN model of a single gene data structure for all gene types. These results also provide a strong argument that the good results obtained on the full state pole balancing problems using negative input concentrations are the result of the FGRN exploiting particularly the sign of the inputs, and not only the additional input granularity.

Table 4.9: Results for the FGRN model with a doubled input concentration range and ALPS GA search on pole balancing problems. The organisation is the same as that of Table 4.5.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	88%	100000	91491	26659	3365	3916	2303
SPB(1.0m)	82%	100000	82936	36511	3660	3540	2072
SPB(2.0m)	68%	100000	69757	44311	3917	4392	2470
DPB	0%	41	47	17	-	-	0
SPB(0.5m) NV	44%	36172	50677	46400	5193	5130	2802
SPB(1.0m) NV	54%	100000	58793	46871	5467	5928	2852
SPB(2.0m) NV	66%	100000	68797	45198	5436	5501	2120
DPB NV	0%	33	35	9	-	-	0

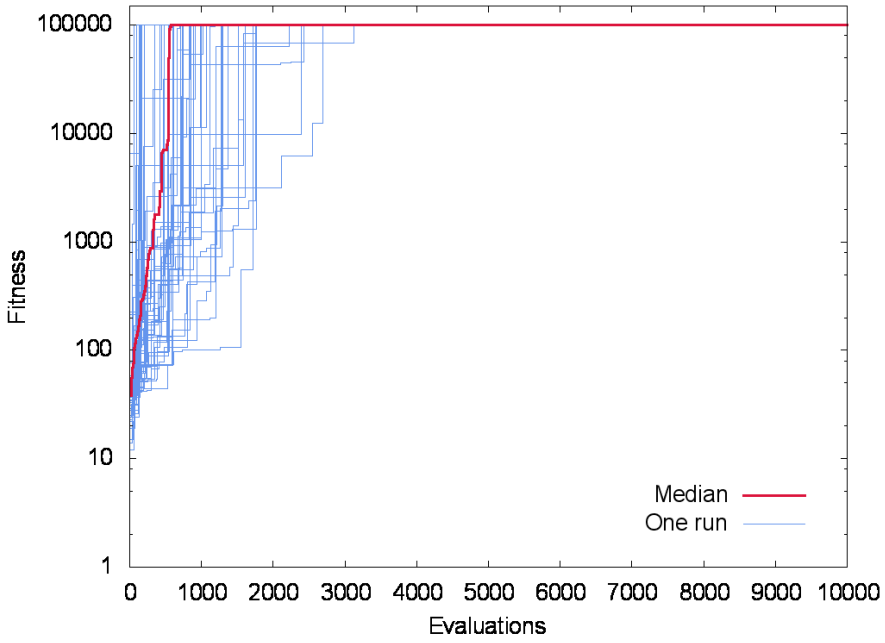


Figure 4.20: FGRN model with negative concentrations on the single pole balancing problem with 1.0m pole (SPB(1.0m)).

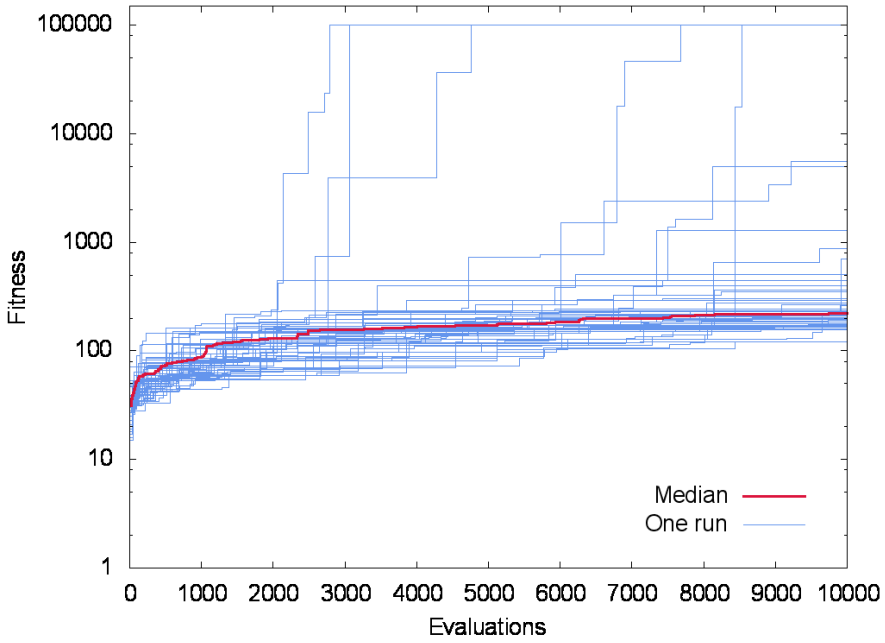


Figure 4.21: FGRN model with negative concentrations on the single pole balancing with 1.0m pole and no velocity inputs (SPB(1.0m) NV).

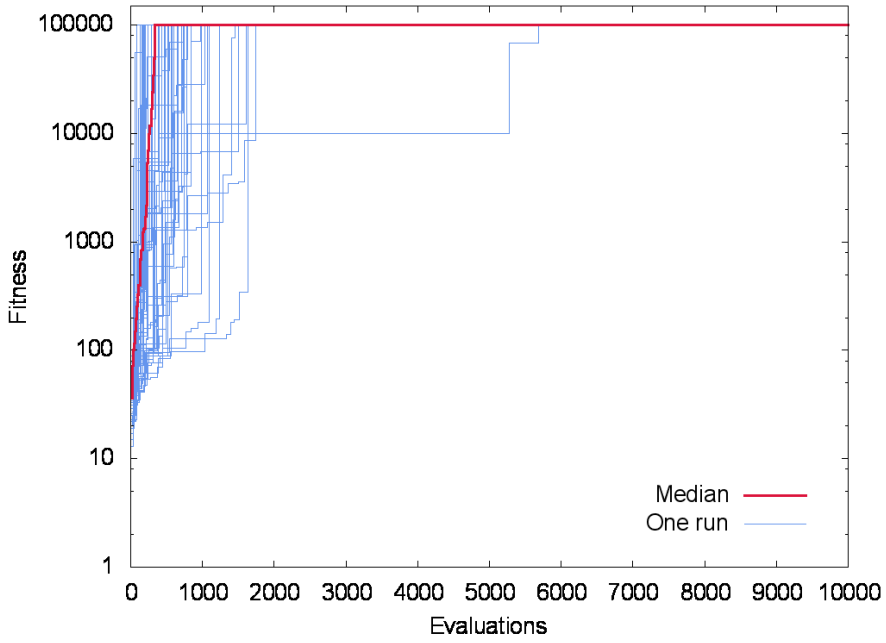


Figure 4.22: FGRN model with negative concentrations on the single pole balancing problem with 0.5m pole (SPB(0.5m)).

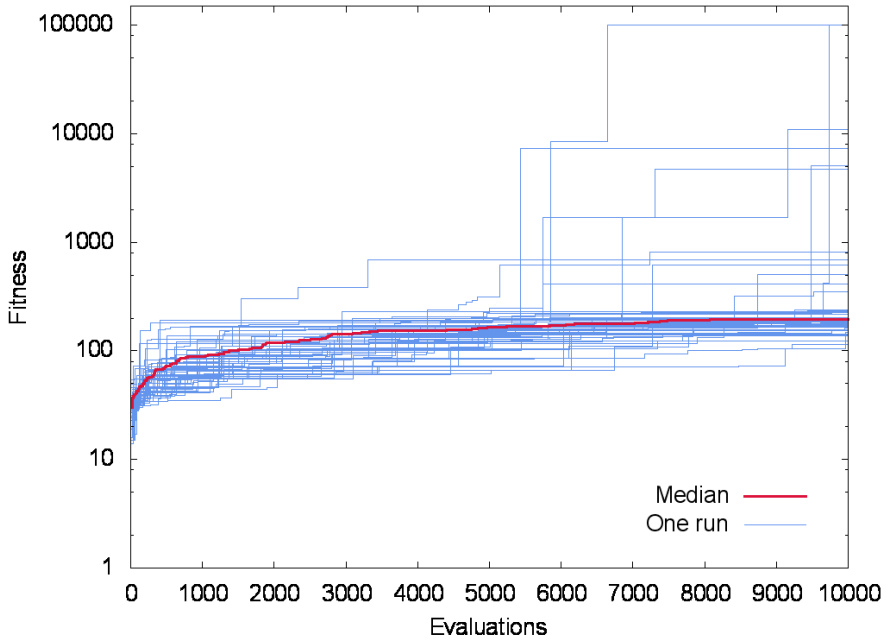


Figure 4.23: FGRN model with negative concentrations on the single pole balancing with 0.5m pole and no velocity inputs (SPB(0.5m) NV).

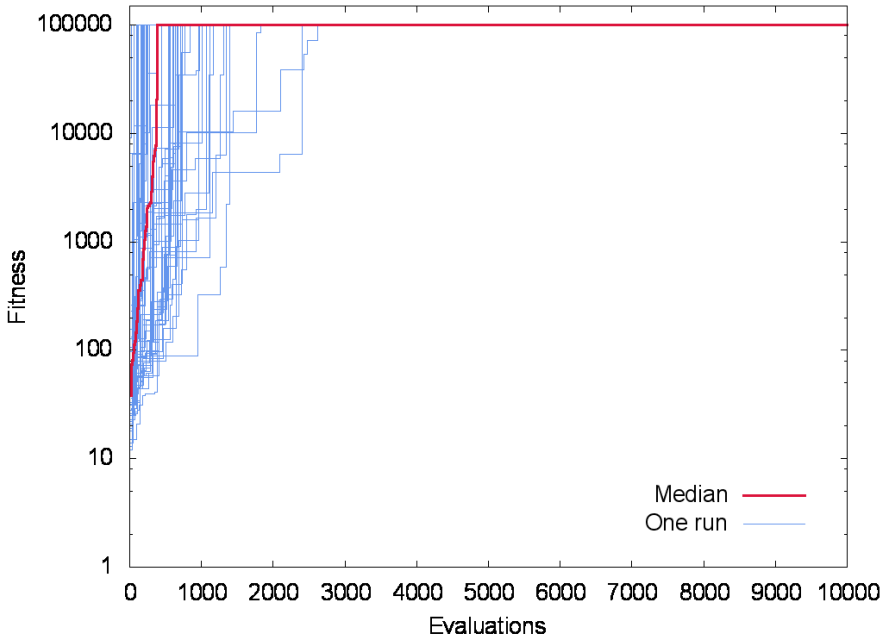


Figure 4.24: FGRN model with negative concentrations on the single pole balancing problem with 2.0m pole (SPB(2.0m)).

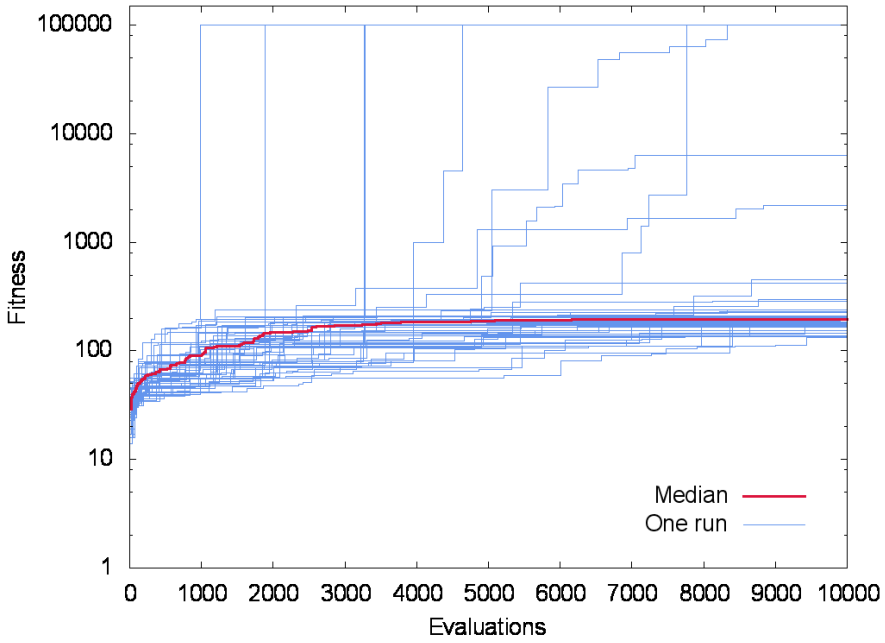


Figure 4.25: FGRN model with negative concentrations on the single pole balancing problem with 2.0m pole and no velocity inputs (SPB(2.0m) NV).

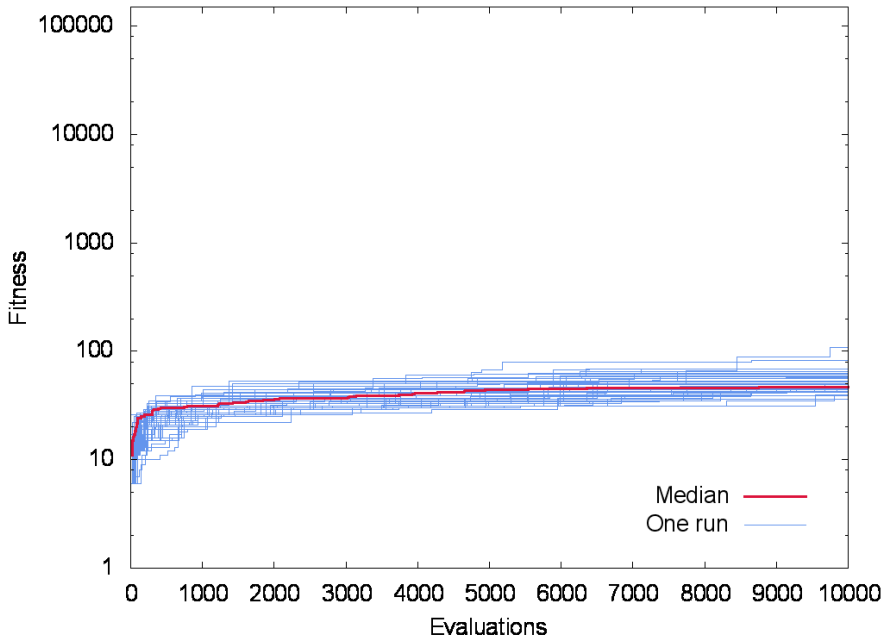


Figure 4.26: FGRN model with negative concentrations on the double pole balancing problem (DPB).

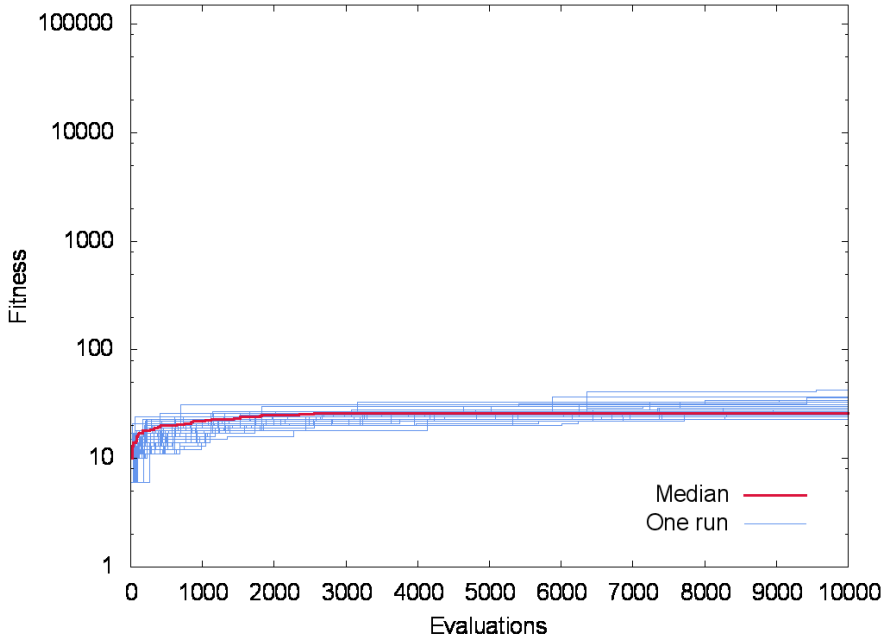


Figure 4.27: FGRN model with negative concentrations on the double pole balancing problem with no velocity inputs (DPB NV).

4.5 Behavioural concentration activation check

Looking at the algorithm of the FGRN main loop (see Algorithm 4.1), it can be seen that the activation of a behavioural gene is a function solely of the gene activation threshold (AT) and of the sum of differences (Δ_P) between the cytoplasm and the gene's promoter protein; the activation of a behavioural gene does not depend on the current concentrations. Accordingly, successful controllers relied only on the presence or absence of regulatory proteins emitted into the cytoplasm at previous timesteps to determine the activation of the behavioural gene. This means the system was unable to rely on inputs from the current timestep to generate its output.

To allow the behavioural genes to take current input into account when determining output, a concentration requirement for behavioural gene activation that was included in Bentley's initial model [Ben04b] but relaxed in further work [Ben05] will be re-enabled. The only change is in the algorithm, in the activation section : when the probabilistic activation condition is fulfilled, an additional condition is added to activation for behavioural genes : the gene is only activated if the mean concentration $\bar{c}$ is greater then the gene's concentration threshold (CT).

This modification is made cumulatively on top of the change to the ALPS GA, and in addition to the negative input concentrations. All experimental settings are kept the same as in the previous section.

Table 4.10: Results for the FGRN model with negative protein concentration, ALPS GA search, and behavioural concentration check on the pole balancing problem. The organisation is the same as that of Table 4.5.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	100%	100000	100000	0	408	559	487
SPB(1.0m)	100%	100000	100000	0	355	438	383
SPB(2.0m)	100%	100000	100000	0	335	502	454
DPB	0%	125	126	17	-	-	0
SPB(0.5m) NV	100%	100000	100000	0	1780	1997	1337
SPB(1.0m) NV	100%	100000	100000	0	2059	2396	1561
SPB(2.0m) NV	100%	100000	100000	0	1906	2179	1515
DPB NV	0%	45	46	8	-	-	0

The pole balancing experiments were run, and this change greatly improved performance. The system also found successful controllers in every run of full-state single pole balancing, but was also able to reliably solve all versions of the partial-state single pole balancing problem with velocities withheld. Additionally the solution to full-state single pole balancing where found more quickly than without the check (on average in under 500 evaluations). This improvement to the system did not however make any difference to the performance on the double pole balancing. As in previous sections, the plots in the following four pages detail the results of the experiments on each variation of the problem.

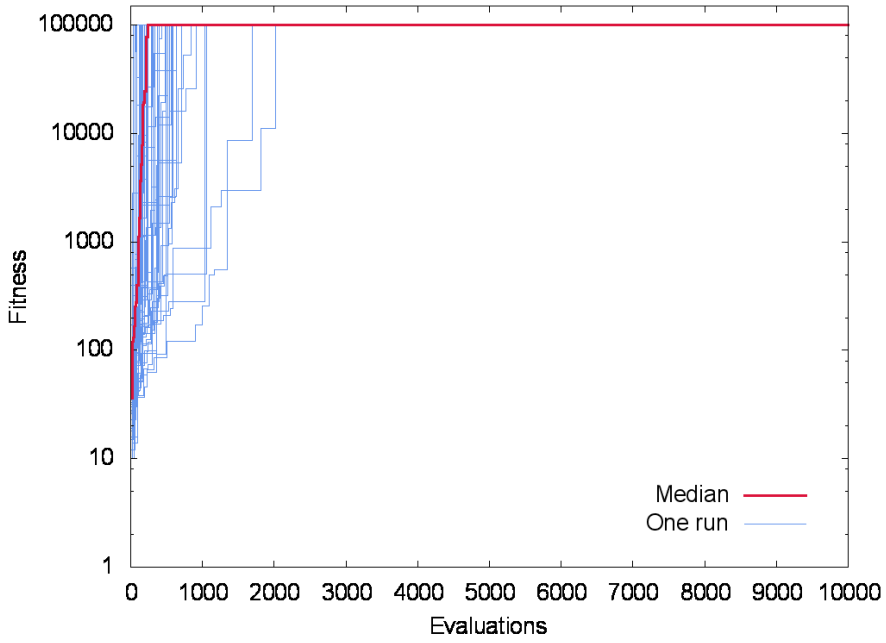


Figure 4.28: FGRN model with behavioural CT check on the single pole balancing problem with 1.0m pole (SPB(1.0m)).

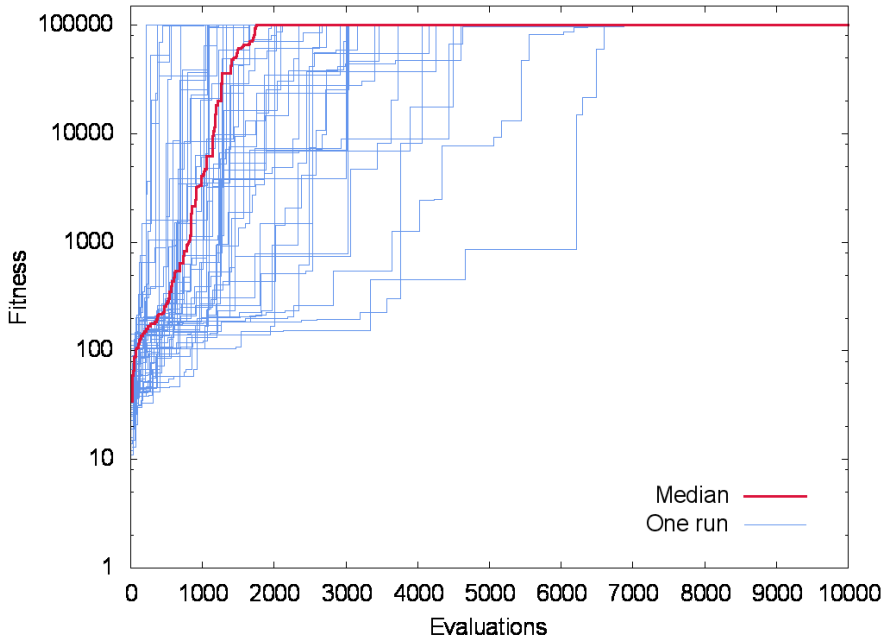


Figure 4.29: FGRN model with behavioural CT check on the single pole balancing problem with 1.0m pole and no velocity inputs (SPB(1.0m) NV).

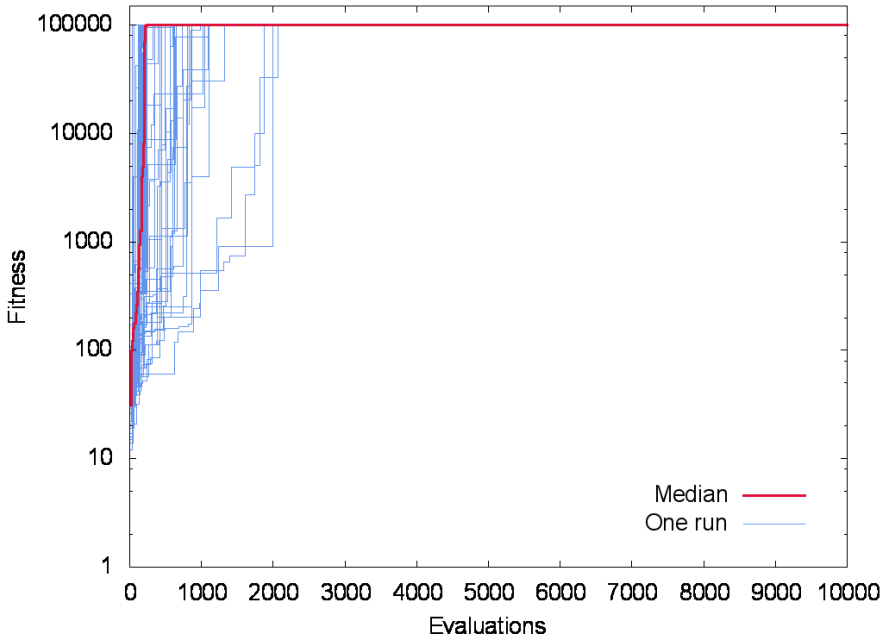


Figure 4.30: FGRN model with behavioural CT check on the single pole balancing problem with 0.5m pole (SPB(0.5m)).

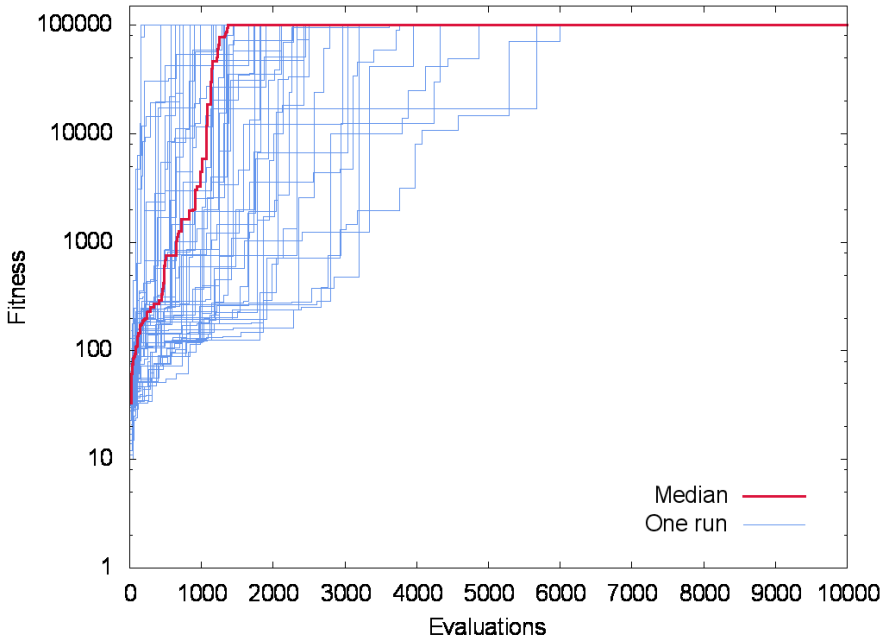


Figure 4.31: FGRN model with behavioural CT check on the single pole balancing problem with 0.5m pole and no velocity inputs (SPB(0.5m) NV).

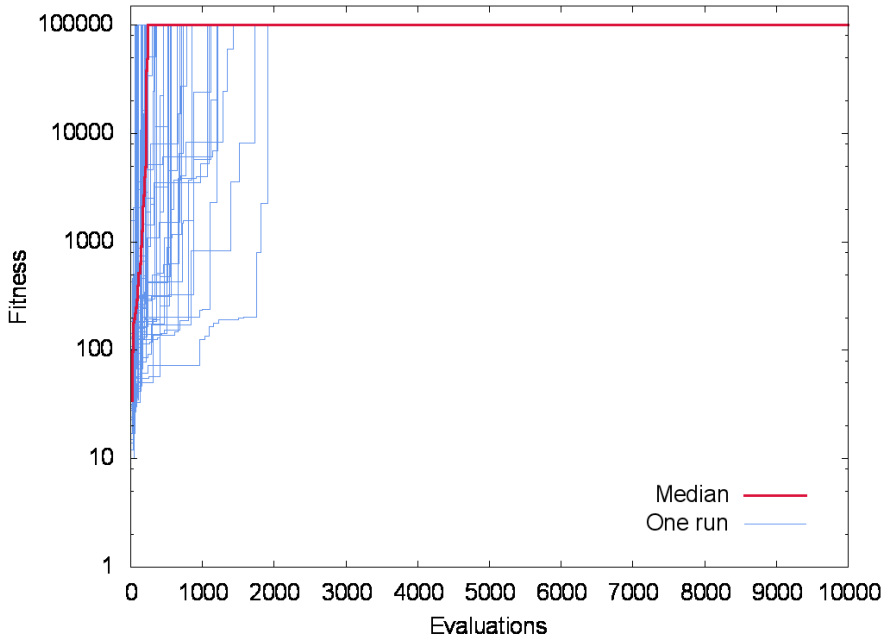


Figure 4.32: FGRN model with behavioural CT check on the single pole balancing with 2.0m pole (SPB(2.0m)).

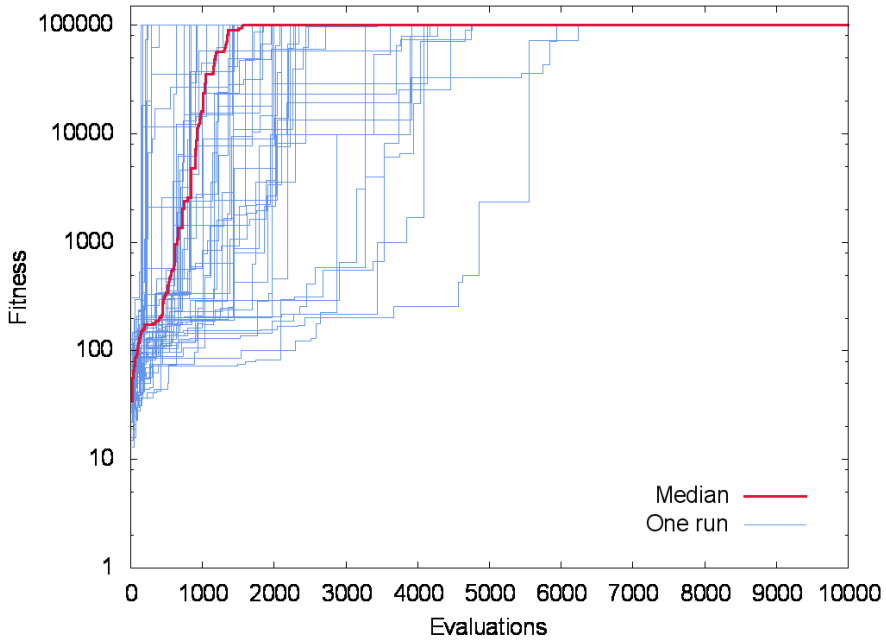


Figure 4.33: FGRN model with behavioural CT check on the single pole balancing problem with 2.0m pole and no velocity input (SPB(2.0m) NV).

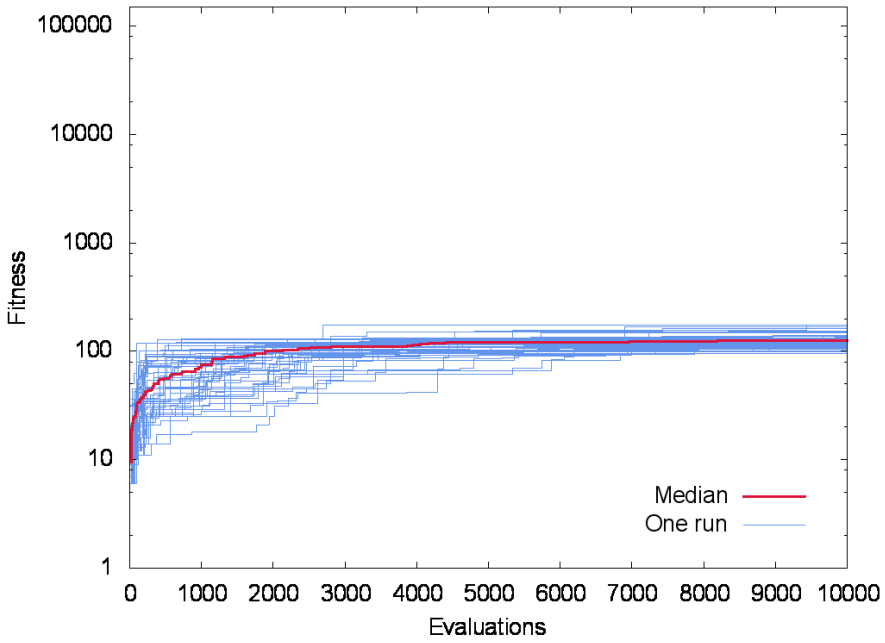


Figure 4.34: FGRN model with behavioural CT check on the double pole balancing problem (DPB).

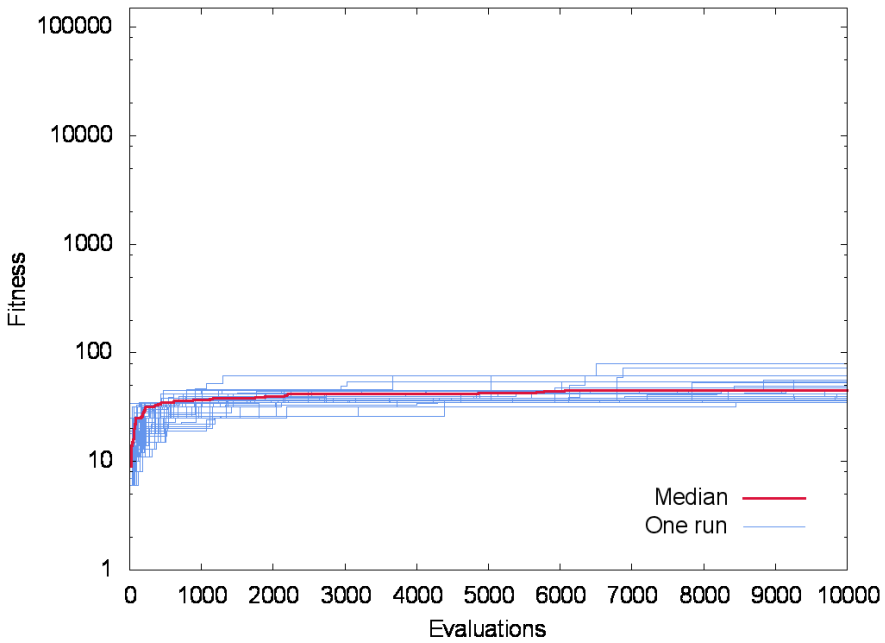


Figure 4.35: FGRN model with behavioural CT check on the double pole balancing problem with no velocity input (DPB NV).

4.6 Performance comparison with a neuroevolution model

Neuroevolution, using evolutionary search methods to evolve the weights and optionally structure of networks of neurons have been used for the generation of controllers for reinforcement learning problems such as the pole balancing. A simple recurrent neural network (RNN) model is implemented to provide reference performances to compare the performances of the FGRN model.

Similarly to the four regulatory and one behavioural genes in the FGRN genomes, the RNN genomes evolved have five neurons, one of which also act as an output neuron. Each neuron has $w = i + n + 1$ weights, where i is the number of inputs, n is the number of neurons, and one extra weight is added as bias. At each time step, the full input vector shared by all neurons is composed of the inputs of the controller (e.g. the pole balancing system's state), the outputs of the neurons at the previous timestep (set to zero for the initial iteration), and the number one for bias. For each neuron, the scalar product of the full input vector and the neuron's weight vector is calculated; the output of the neuron is then the hyperbolic tangent of the resulting weighted sum.

Weights in the initial genomes are uniformly generated within the range $[-1, 1]$. When a weight is mutated a random value uniformly distributed in the range $[-0.5, 0.5]$ is added to the weight. Each weight of a genome produced by crossover is picked randomly from one of the two parent genomes.

Experiments are run, RNN genomes being evolved via ALPS, with the same settings as in the previous section. The same scaling of the controller's input is applied as for FGRN experiments. As in previous sections, the plots in the following four pages detail the results of the experiments on each variation of the pole balancing problem.

Table 4.11: Results for RNN on the pole balancing problems, with ALPS GA search. The organisation is the same as that of Table 4.5.

	%	Final fitness			Evaluations to success		
		Median	Mean	SD	Median	Mean	SD
SPB(0.5m)	100%	100000	100000	0	218	314	296
SPB(1.0m)	100%	100000	100000	0	292	371	258
SPB(2.0m)	100%	100000	100000	0	378	530	487
DPB	12%	300	12728	32279	7180	6516	1986
SPB(0.5m) NV	100%	100000	100000	0	437	592	530
SPB(1.0m) NV	100%	100000	100000	0	481	588	434
SPB(2.0m) NV	100%	100000	100000	0	682	772	621
DPB NV	0%	243	272	159	-	-	0

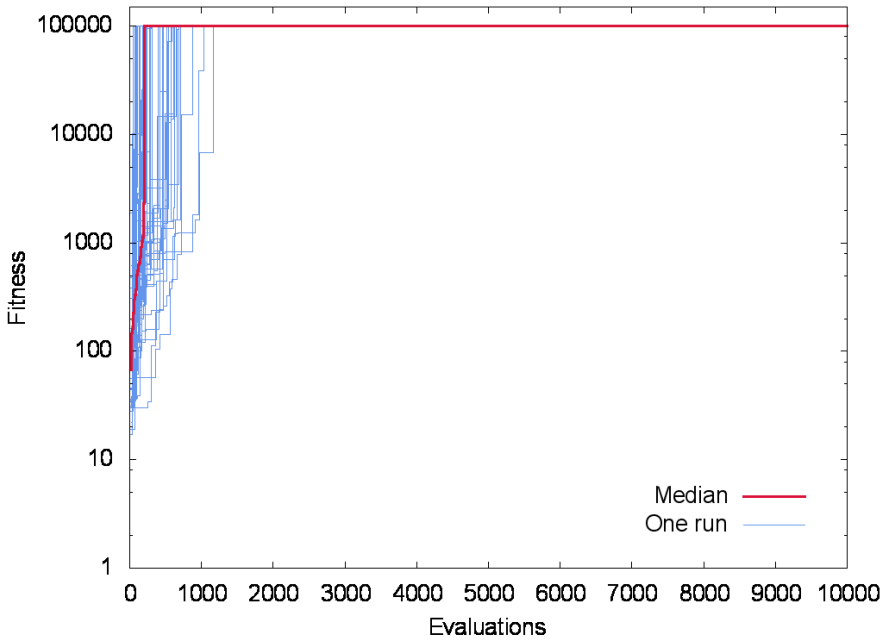


Figure 4.36: RNN on the single pole balancing problem with 1.0m pole (SPB(1.0m)).

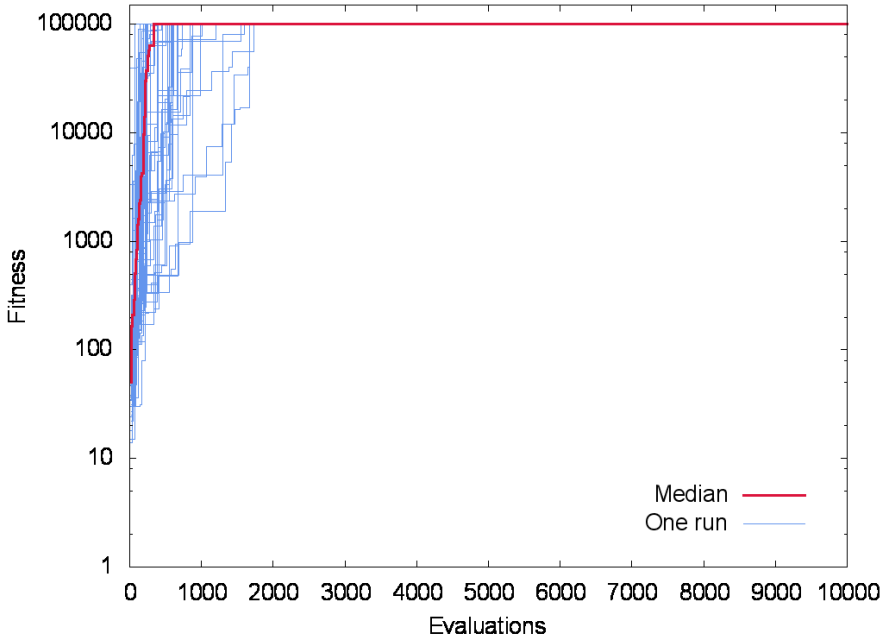


Figure 4.37: RNN on the single pole balancing problem with 1.0m pole and no velocity inputs (SPB(1.0m) NV).

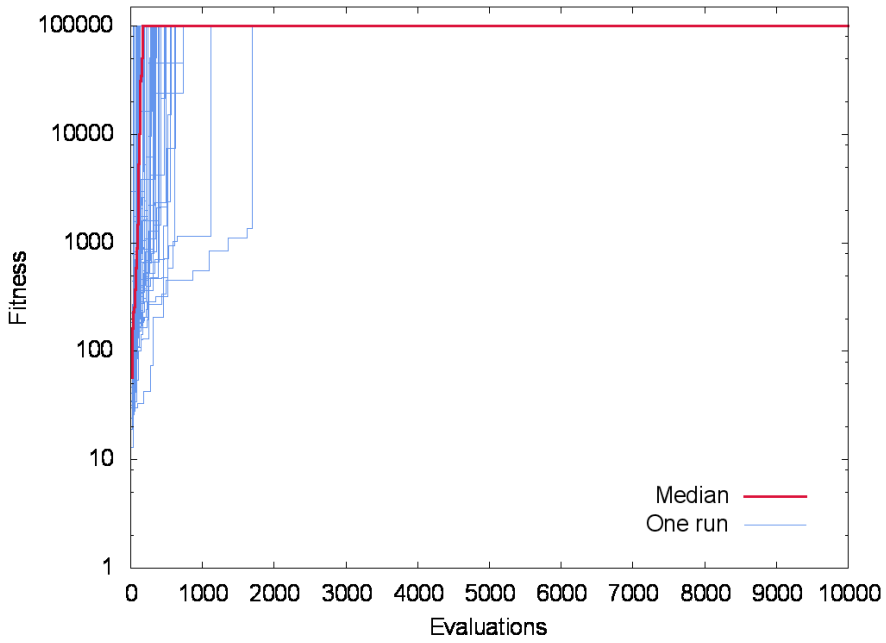


Figure 4.38: RNN on the single pole balancing problem with 0.5m pole (SPB(0.5m)).

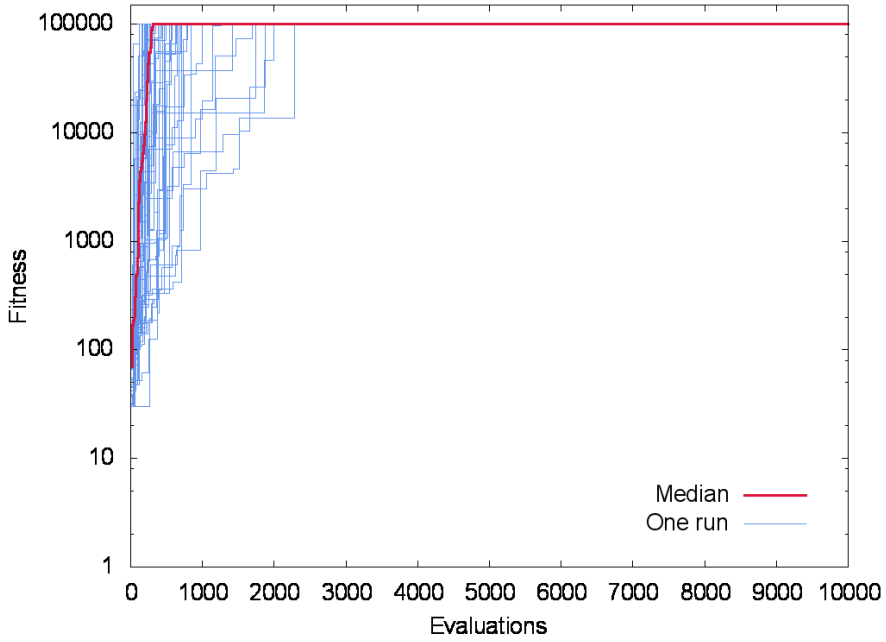


Figure 4.39: RNN on the single pole balancing problem with 0.5m pole and no velocity inputs (SPB(0.5m) NV).

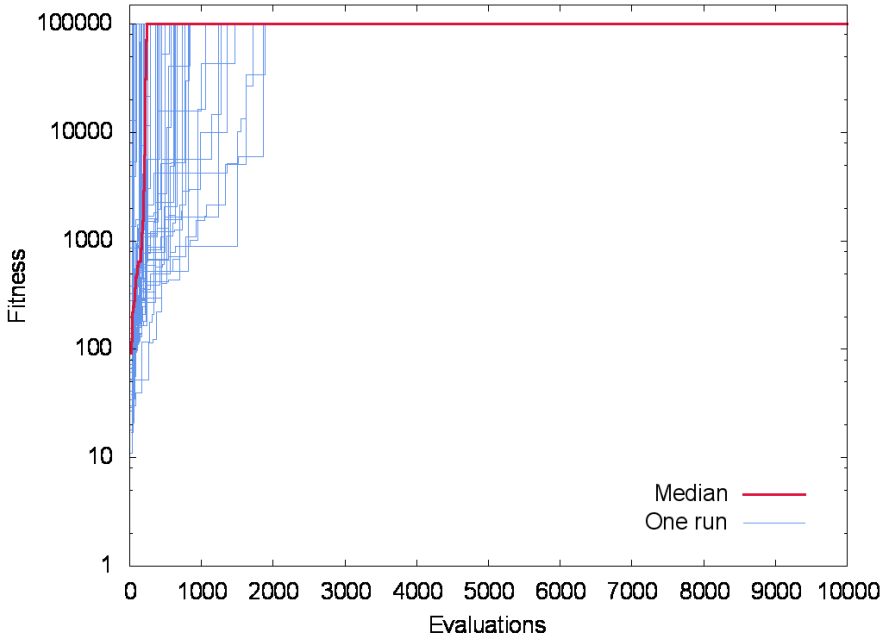


Figure 4.40: RNN on the single pole balancing problem with 2.0m pole (SPB(2.0m)).

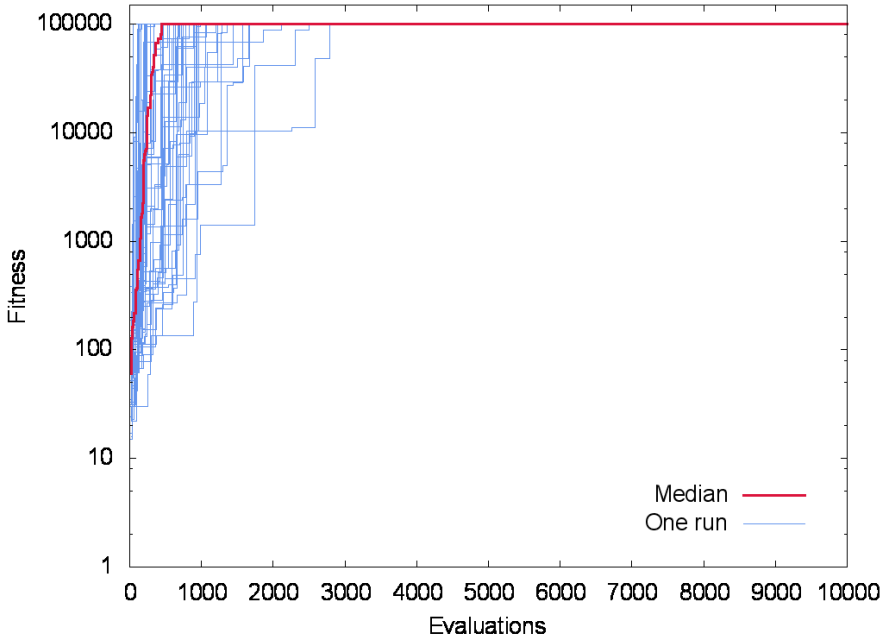


Figure 4.41: RNN on the single pole balancing with 2.0m pole and no velocity inputs (SPB(2.0m) NV).

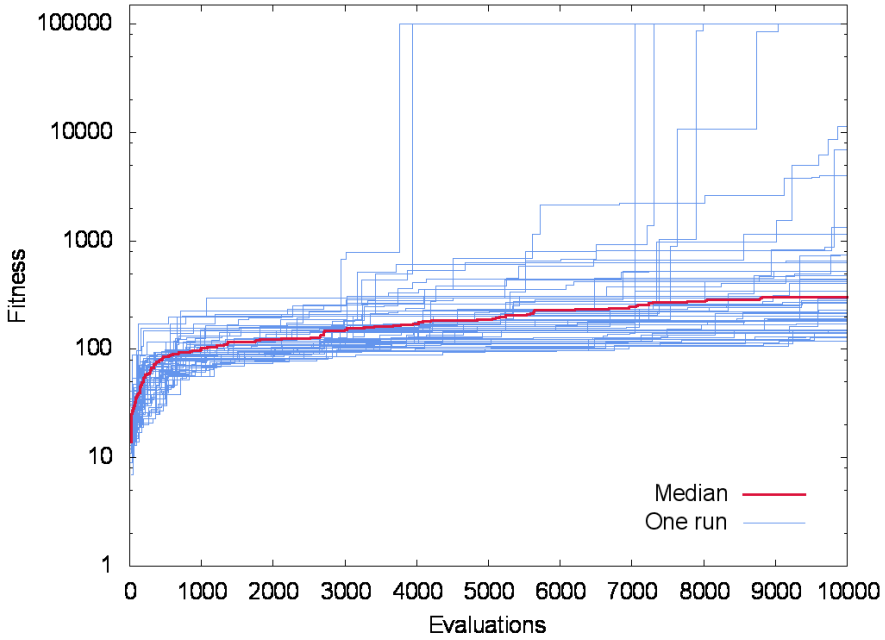


Figure 4.42: RNN on the double pole balancing problem (DPB).

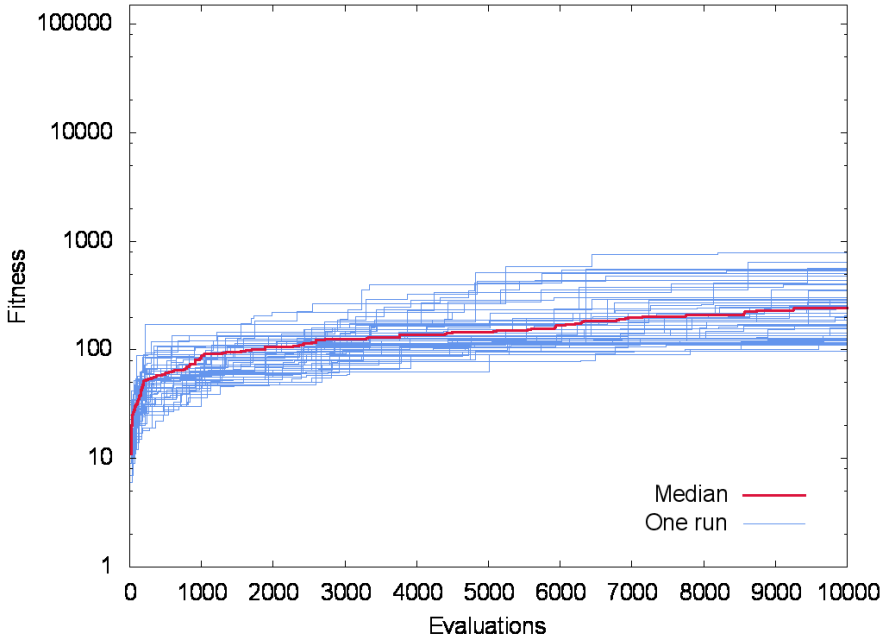


Figure 4.43: RNN on the double pole balancing problem with no velocity input (DPB NV).

Overall the performance of the RNN model on all versions of single pole balancing was superior to the best performance obtained so far with the FGRN model. Additionally, some successful controllers were found for full-state double pole balancing. As mentioned in the literature review, Section 2.1.2, neuroevolution systems exist that can solve all the versions of the pole balancing studied here; but the fact that the basic, unoptimised RNN model used here outperformed the FGRN model, not only on single pole balancing, but also in achieving some success on the double pole is a strong indication further improvements to the FGRN model are needed.

4.7 Discussion

In this chapter, the adaptations required for the application of the FGRN model were described in detail. The FGRN model was then successfully applied to the full-state single pole balancing problem (a standard, well-recognised control benchmark problem), with however initially limited reliability, the system being only able to find a successful controller in some of the runs. Changing the GA used in the system from Bentley's FGA to the ALPS GA led to both an increase in the reliability with which successful controllers were found, but also to a decrease in the computational costs. A further improvement, allowing negative inputs to be encoded as negative concentrations led to successful controllers being reliably generated for every run of the full-state single pole balancing problems; the mean number of evaluations required until a successful controller is found was also significantly reduced. A change in the FGRN algorithm reduced that number of evaluations further; it also led to complete, reliable success on the partial-state single pole balancing problems, for which so far the generation of successful controllers had been very unreliable. However throughout all these improvements, no successful controller was generated for double pole balancing.

For comparison with neuroevolution, a basic recurrent neural network (RNN) model was implemented and tested on the same pole balancing problems. The RNN model performed better than the FGRN model on the single pole balancing problems in terms of the number of evaluations required to find a successful controller, and more importantly was able to solve the full-state double pole balancing problem on some of the runs. Additionally more sophisticated neuroevolution systems exist that can reliably solve the double pole balancing problem.

As can be seen in the literature review, Table 2.3 The FGRN model's performance on single pole balancing are on a par with other evolutionary methods (including some neuroevolution methods), and does better than traditional reinforcement learning, value surface based methods. However the complete failure of the system to generate any successful controller for the double pole balancing problem where even a basic, unoptimised neuroevolution method shows some success is unsatisfactory.

To improve the FGRN model further, the next chapter will look more deeply into the FGRN model's workings, by investigating what has been often put forward as the model's main specificity [Ben04b] [Ben09] : the protein encoding process in which fractal proteins are produced from the genetic protein definition.

Chapter 5

Investigating Protein Encoding

The results obtained in the previous chapter on the pole balancing control problem are interesting, being one of the first applications of a GRN model to a real control problem, and are a vast improvement on pure reinforcement learning approaches. But they are still trailing behind other approaches, including neuroevolution. Some improvements to the FGRN model are needed.

In the FGRN model, each gene contains two protein definitions, respectively encoding a promoter protein and an output protein. The promoter protein determines the condition under which a gene is activated. This is a big departure from biology in which there is no equivalent of a promoter protein as such. In biology that role is mostly taken by the gene's cis-regulatory region, which does not code for anything, but interacts directly with regulatory proteins (also known as transcription factors). The output protein is produced into the cytoplasm on regulatory gene activation.

An FGRN encoding method translates a protein definition in a gene into a greyscale bitmap; e.g. for fractal proteins, from an $< x, y, z >$ triplet it produces a rectangular greyscale bitmap picture of a portion of the Mandelbrot fractal. The FGRN model's chemistry operations (mask, merge, promoter comparison, detailed in Section 3.1.4) operate on a pixel-by-pixel basis, with no neighbourhood effect. The chemistry operations can therefore be applied to arbitrary 15×15 greyscale bitmaps with pixel values in the range $[0, 127]$, and not only to the products of fractal sampling. This makes it possible to test different protein encoding methods while keeping exactly identical all other components of the system; the ease with which suitable GRNs are found by a genetic algorithm to solve a given problem (i.e. the evolvability of the model for that given problem) will be shown to be greatly affected by the encoding. It would be possible to directly define, without encoding, 15×15 protein bitmaps within the genes; but the much bigger size of the resulting genomes would be extremely detrimental to the performance of the evolutionary search process.

This chapter focuses solely on investigating and improving the protein definition and encoding in the FGRN model, in terms of expressivity and computational cost. The fractal nature of this component has often been cited as the source of the evolvability of the system, and was seen as important enough to name the whole system (it was initially introduced by Bentley as simply "Fractal Proteins" [Ben04b]).

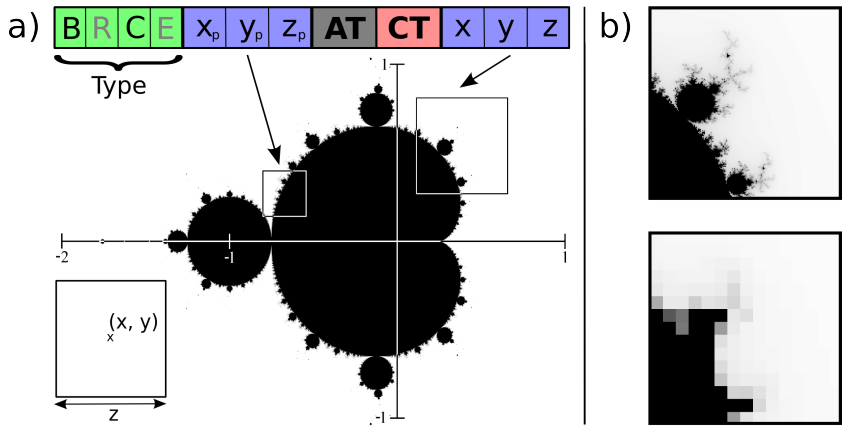


Figure 5.1: Fractal protein encoding. The promoter $< x_p, y_p, z_p >$ and output $< x, y, z >$ protein coordinates define different fractal portions of the Mandelbrot set. Bottom left: a schema describing the mapping from protein coordinates to fractal portion. b) Top: the fractal portion pointed at by the output coordinates $< x, y, z >$ of the gene. Bottom: the resulting $< 15 \times 15 >$ fractal protein bitmap. (Reproduction of Figure 3.6 for convenience.)

As a consequence, there has been little work investigating the FGRN system without fractals; Bentley experimented with removing fully the generative protein encoding step, the coordinates $< x, y, z >$ being then used directly as proteins and promoters by the running system [Ben05], but did not look at alternative protein encodings. In that work, Bentley found the use of fractal protein encoding improved the quality of solutions found on the problem of approximating the square root function over a limited range, compared with the same FGRN system without a generative protein encoding step. Little detail is however given on the actual workings of the protein-less system. Bentley attributed the difference of performance to greater evolvability resulting from the higher number of possible interactions between fractal proteins than between the $< x, y, z >$ coordinates used to generate them, resulting in a system better able to not get stuck in of local optima [Ben05]. Bentley's results are a good argument for the use of a generative protein encoding step in the system, but say comparatively little about the suitability of fractals as protein encoding; could simpler, more effective, protein encodings exist?

In this chapter, first the fractal protein component (the combination of definition and definition-to-bitmap) will be discussed, and specific limitations identified. Subsequently two alternative protein encodings, each mitigating one of these limitations, will be introduced:

- Each protein produced by fractal encoding being a different view of the same fractal, the resulting fractal proteins are limited in the basic shapes they can take. In order to enlarge the space of available shapes 'Mondrian' proteins are introduced, allowing for the generation of proteins with multiple solid black areas, and a much wider range of basic shapes than fractal proteins.

- On the basis that fractal proteins under-exploit the greyscale, almost all pixels being either black, or very close to white, 'landscape' proteins are introduced, providing smooth gradients covering the full greyscale, while still possessing one significant solid black area.

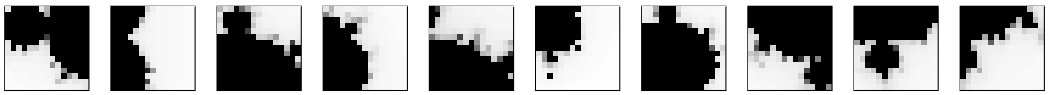


Figure 5.2: The ten pre-evolved fractal proteins used in FGRN random genome initialisation. (Reproduction of Figure 3.5 for convenience.)

The performances of these novel protein representations are then compared to those of fractal proteins via computational experiments on the pole balancing, and the generation of activation patterns as in Section 3.2. A statistical analysis of large random samples of proteins produced by each of the protein encodings is then provided. A discussion follows as to how to incorporate the identified desirable characteristics within the same system.

5.1 Critique of fractal protein encoding

As seen in Section 3.1.3, a fractal protein bitmap is generated from a triplet of real coordinates $< x, y, z >$: $< x, y >$ defines the centre of the square portion of the fractal portion to sample, and z is the side of that square portion; the protein bitmap is then generated from the square portion, by sampling at each point of a 15×15 grid: the result is a 15×15 greyscale bitmap with integer values in the range $[0, 127]$ (see Figure 5.1). According to Bentley, the aim of using fractal proteins was to provide the system with "fractal shapes of infinite complexity" [Ben09]. However, observing the 15×15 greyscale bitmaps that are the end product of the generative process of a fractal protein, the large majority of fractal proteins fit the description of a single black blob on a white background with a thin border of grey. This can be seen in the ten pre-evolved proteins which form the building blocks of the genomes in the initial random population in a FGRN run (see Figure 5.2).

From a consideration of the workings of the system, and for evolvability purposes, it is sensible that the protein bitmap be composed of a number of solid black regions, and of some continuous regions of other grey values. Black portions are essential for masking with the promoter and receptor proteins, and having solid colours or gradients in the rest of a protein is desirable so that a small change to a gene's protein coordinate or affinity threshold results in a similarly small change in the behaviour of the gene. However, the limited range of possible general shape the protein bitmaps take is unlikely to be desirable, as it prevents the emergence of shapes which are composed of more than one solid black area. Mondrian proteins, introduced below, illustrate a simple kind of general shape unobtainable with fractal proteins (see Figure 5.4). Such protein bitmaps, used in the role of a promoter protein, would make it possible to restrict more precisely the area or areas of the cytoplasm used to determine gene activation.

Another limitation of fractal proteins is their narrow use of the greyscale middle range; most parts of a fractal protein are either black, or almost white (as can be seen in Figure 5.2). A better exploitation of the greyscale range through gradients in fractal proteins could allow for a smoother mapping between mutations in the protein definition and the resulting protein. The landscape protein encoding, introduced in the next section, aims to achieve this.

Regarding the evolvability of fractal proteins, given the mutation range ($[-0.5, +0.5]$) for the fractal

coordinates used in this work and Bentley's previous work, proteins from mutated fractal coordinates show little similarity with the proteins obtained from the starting definition, making it unlikely that fractal encoding provides a significant advantage in terms of evolvability. Preliminary experiments on π approximation showed decreased performances when attempting to make mutated fractals more similar to their parent via the use of smaller mutation ranges.

5.2 Mondrian protein encoding

To allow more expressivity in the protein shape than fractal encoding can provide, Mondrian proteins are introduced, and named for their superficial resemblance to the work of Dutch painter Piet Mondrian (see Figure 5.3). The aim of Mondrian encoding is to maximise the diversity of proteins produced, allowing for shapes similar to those obtained with fractal encoding, and providing additional ones, but also keeping the data size of the definition to a minimum in order to facilitate exploration of the parameter space.

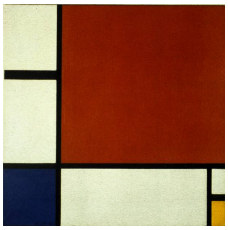


Figure 5.3: Piet Mondrian's Composition II in Red, Blue, and Yellow.

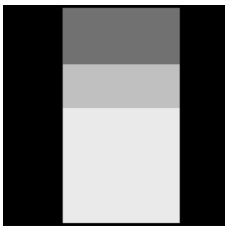


Figure 5.4: An example Mondrian protein, created from two merged Mondrian portions. This type of layout of the black parts is hard to obtain with fractal proteins.

A Mondrian protein is composed of one or several merged Mondrian 'portions'. A Mondrian portion is a protein sized (15×15) two-dimensional bitmap divided into three regions, each with a constant value of grey. The portion bitmap is first split vertically or horizontally into two possibly unequal regions, the resulting sections being each set to different, specified grey values. A black band is then added perpendicularly, on a side of the bitmap.

A Mondrian portion is defined by the following parameters (each portion being defined in only three bytes):

- The direction d of the initial split (1 bit), horizontal or vertical.
- The position p of the initial split (4 bits, in $[0, 15]$).

- Two grey values v_1 and v_2 (7 bits each, in $[0, 127]$) defining the grey values of each section.

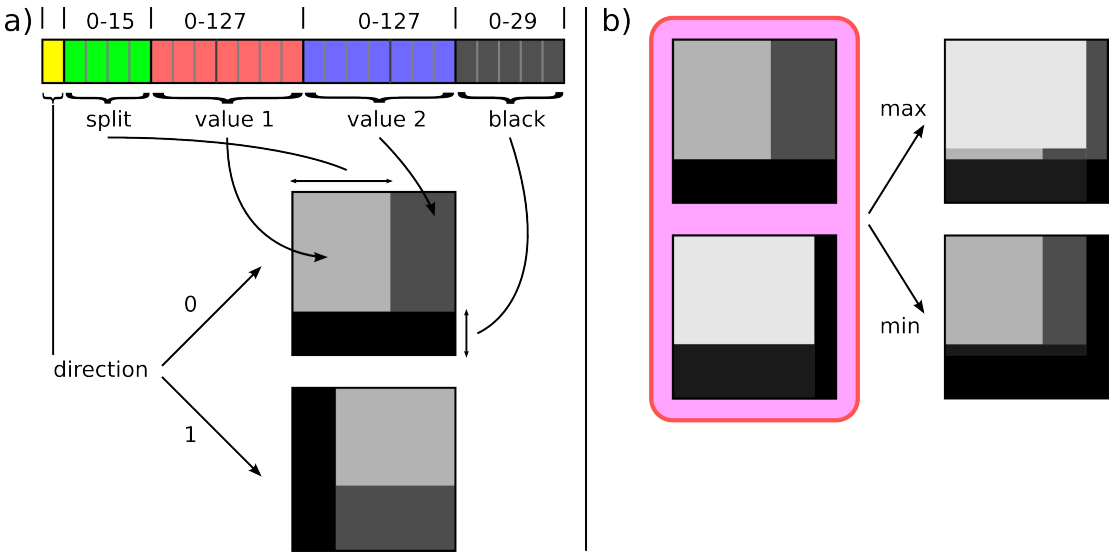


Figure 5.5: Mondrian portions and proteins. a) The encoding from genetic material to a Mondrian portion. b) A Mondrian protein can be composed from multiple Mondrian portions, merging them by applying a maximum or minimum operation at each pixel of the bitmap.

- A 'black' parameter p_b (5 bits, in $[0, 29]$) defining both the location of the black band and its width. For instance, for an horizontal black band, a value n less than or equal to 15 creates a black band on the first n lines of the portion bitmap, whereas for a value greater than 15 it creates a black band on the last $n - 15$ lines of the bitmap.

A Mondrian protein containing more than one portion is generated by taking the maximum or minimum value at each pixel; this allows the black portion of a Mondrian protein composed of multiple portions to take a multitude of shapes, some of which are not obtainable via fractal encoding (see for example the protein in Figure 5.4). Figure 5.5 illustrates the process of generating a Mondrian protein. The initialisation and mutation ranges for each of a portion's parameters are shown in Table 5.1; as for fractal protein definitions, crossover occurs through parameter swapping (each parameter is taken at random from one of the two parents).

Table 5.1: The full, initialisation, and mutation ranges for the parameters of a Mondrian protein definition. The initialisation and mutation values are taken from a uniform distribution. Mutated parameters are then constrained to stay within the 'full' range.

Parameter	**Full**	**Initialisation**	**Mutation**
p	$[0, 15]$	$[0, 15]$	$[-8, +8]$
v_1, v_2	$[0, 127]$	$[0, 127]$	$[-64, +64]$
p_b	$[0, 29]$	$[0, 29]$	$[-15, +15]$

5.3 Landscape protein encoding

The landscape protein encoding is an even simpler encoding that however aims to create proteins which take full advantage of the greyscale range available. As discussed at the beginning of this chapter, all protein chemistry operations in the FGRN model act on a pixel-by-pixel basis, with no neighbourhood effects. One implication of this is that protein bitmaps can be generated independently to the workings of a running FGRN system. They can take the form of a 15×15 two-dimensional bitmap (as is the case for fractal and Mondrian proteins) or a one-dimensional array of length 225 ($= 15 \times 15$). The latter is the approach taken with landscape protein encoding.

First, the landscape protein definition sets two positions in the one-dimensional array for which a 'height' is given; the height values at all other positions of the array are linearly interpolated to provide a gradient between the two set positions. A flat valley of height zero (the equivalent of the black part in fractal and Mondrian proteins) is then carved out around another given position of the array.

Table 5.2: The full, initialisation, and mutation ranges for the parameters of a landscape protein definition. The initialisation and mutation values are taken from a uniform distribution. Mutated parameters are then constrained to stay within the 'full' range (as in Mondrian proteins).

Parameter	**Full**	**Initialisation**	**Mutation**
p_1, p_2, p_b	$[0, 224]$	$[0, 224]$	$[-112, +112]$
h_1, h_2,	$[0, 127]$	$[0, 127]$	$[-64, +64]$
w_b	$[0, 224]$	$[0, 157]$	$[-56, +56]$

The process of generating a landscape protein from the genome definition is illustrated Figure 5.6. Specifically, each landscape protein definition is composed of the following parameters, all subject to mutation:

- Two positions p_1 and p_2 (8 bits each, in $[0, 224]$) defining the indices within the one dimensional array at which the 'height' values are set.
- Two height values h_1 and h_2 (7 bits each, in $[0, 127]$) defining the 'height' value at each of the two positions.
- The position of the centre of the black area p_b (8 bits, in $[0, 224]$).
- The width of the black area w_b (8 bits, in $[0, 224]$).

As with fractal and Mondrian protein definitions, crossover occurs via parameter swap; the initialisation and mutation ranges for each parameter are shown in Table 5.2.

Figure 5.7 displays examples of landscape proteins. The total size of a landscape protein definition is less than 6 bytes, equivalent to the size of a two-portion Mondrian portion, to be contrasted with the total 24 bytes of the three double floating point values which form the definition of a fractal protein.

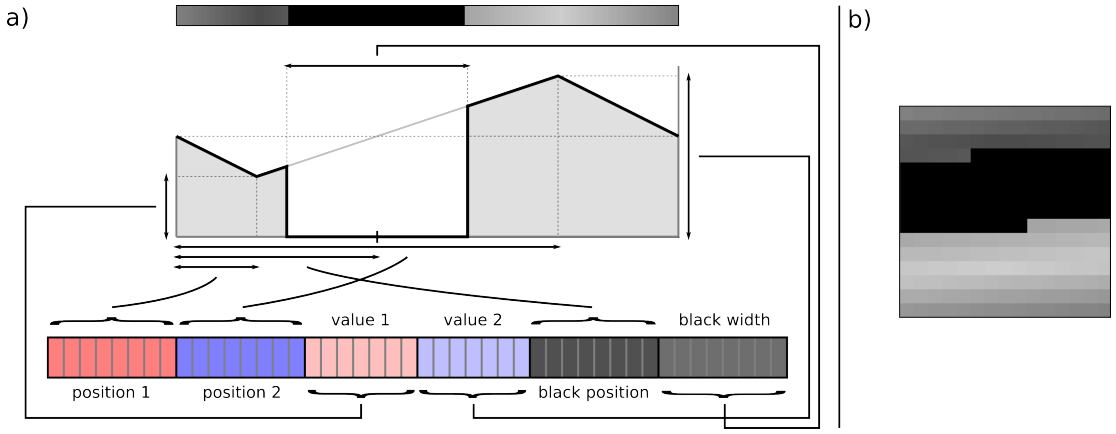


Figure 5.6: Landscape protein encoding. a) The mapping of the five bytes long genetic material below, to the resulting landscape protein above. b) The same protein, represented as a two-dimensional bitmap of same dimension as a fractal or Mondrian protein, for comparison.

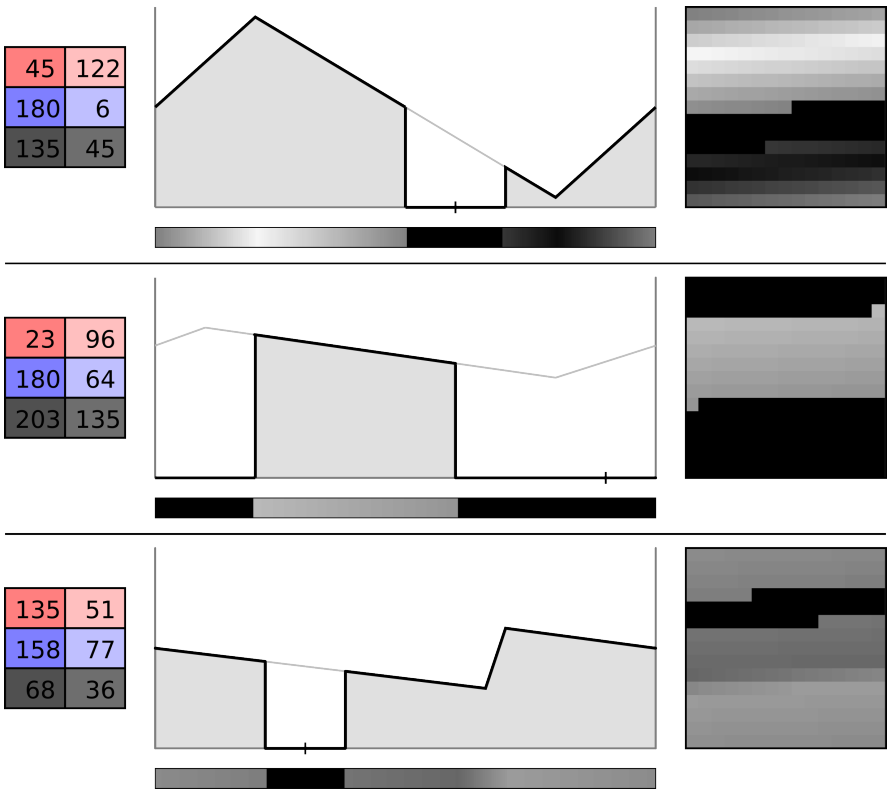


Figure 5.7: Landscape protein examples, illustrating the protein construction, the resulting 225 pixel one-dimensional array, and the corresponding 15×15 bitmap. The array to the left shows in each case the value of the definition parameters; colours match the positions in Figure 5.6.

5.4 Experiments

The fractal nature of the protein encoding was presented as the most important part of the FGRN model [Ben09]. This hypothesis has been challenged in this chapter, by pointing out limitations in the expressivity and evolvability of the resulting proteins, and introducing alternative encodings which display greater expressivity in terms of both exploitation of the grey scale and the diversity of shapes produced; additionally these new encodings do not require the use of pre-evolved protein coordinates to initialise the genomes of the starting population. In this section, the new protein encodings are tested against the fractal encoding on control and developmental problems.

The control problems are the standard full state single (with velocities) pole balancing problem with a one meter pole, and the standard double pole balancing. The FGRN system (note that this is here defined to not include the protein encoding mechanism) and associated GA are the best performing in the previous chapter, as described in Section 4.5. Note that the whole FGRN system was optimised for use with fractal encoding (in relation to the mutation rate and constants in the activation and protein production functions), which should provide fractal encoding with an advantage. The protein encodings tested are the fractal, landscape, and Mondrian (with one, two and four portions) encodings.

The developmental problems are the generation of each of the activation patterns specified in Section 3.2. Patterns one and two are defined in Figure 3.8 and each require two outputs and therefore two behavioural genes. Pattern three is defined in Figure 3.9 and requires one output, and therefore one behavioural gene. These patterns were used by Bentley for the initial FGRN testing [Ben04b]. The fitness function used is the opposite of the sum of non-matching activations. These developmental problems do not require any input, so only one environmental gene is used, with fixed saturation concentration. With the exception of the double pole balancing, for which no successful controller was found with any of the systems tested, the results of these experiments are detailed in Table 5.3.

Table 5.3: Results for different protein encodings for the full state single pole balancing problem (SPB) and the activation pattern problems over fifty runs for each experiment. The two numbers represent the mean number of evaluations required to obtain a successful solution to the problem, and in parenthesis the associated standard deviation. If a percentage is included, it describes the proportion of runs which yielded a successful controller (100% is implied otherwise). Mondrian-N refers to a Mondrian protein encoding merging N Mondrian portions.

Encoding	**SPB**	**Pattern 1**	**Pattern 2**	**Pattern 3**
Fractal	438(383)	3703(3838)	1475(1346)	26% - 12012(4118)
Landscape	326(348)	3663(3948)	1125(1128)	30% - 12782(5123)[1]
Mondrian-1	431(349)	94% - 5317(3954)	1120(944)	30% - 9754(5123)[1]
Mondrian-2	534(404)	94% - 6782(4710)	2072(1843)	28% - 9392(4751)
Mondrian-4	980(696)	86% - 8690(4873)	98% - 2343(1986)	16% - 10847(2741)

[1] The identical standard deviations are purely coincidental.

It can be seen that the use of the landscape protein encoding led to improvements in performance on every one of the problems tested compared to the same system using fractal protein encoding. Looking at the overall results the system using landscape protein encoding performed significantly better than the one using fractal protein encoding ($p < 0.03$). The results of the system using Mondrian protein encoding with a single portion were similar to those of the fractal protein encoding alternative, doing better on some and worst and others.

Using Mondrian encoding with a higher number of portions lead to clear decrease in performance; this is mostly attributed to the simple minimum (and respectively maximum) pixel-level operations used to combine multiple portions into one protein which tend to result in proteins that are mostly or even completely black (respectively mostly non-black). Fully black proteins having no effect on the cytoplasm, and fully non-black proteins affecting the whole cytoplasm, they depart from the combination of a black shape with a clear area characterising most proteins produced by the fractal and landscape protein encodings, as well as the Mondrian protein encodings with only one or two portions. Additionally the mutation rate (optimised for the fractal encoding) might be too high for the high number of parameters multi-portion Mondrian protein definitions contain. These issues could be mitigated by combining portions with more elaborate operators than a single min/max, and reducing the mutation rate. However the motivation for doing this is not strong since evidence suggests that the simpler landscape protein encoding should be preferred. It is certainly the case that the system using fractal protein encoding does not distinguish itself despite the settings being tuned for its use.

5.5 Protein statistical analysis

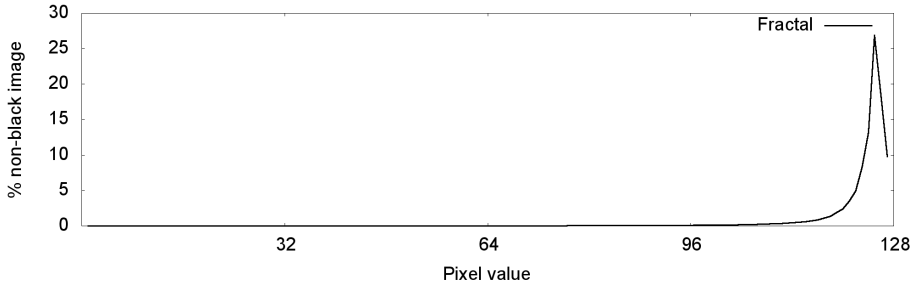
To analyse the statistical properties of each protein encoding, a hundred thousand proteins were generated for each protein encoding. The landscape and Mondrian proteins were generated as detailed in their respective sections above. Fractal proteins used in the FGRN model come from a small set of ten pre-evolved proteins; here random fractal proteins were generated by uniformly taking the $< x, y, z >$ coordinates from the space $[-2, 1] \times [-1, 1] \times [-2, 2]$, aiming to enclose tightly the Mandelbrot set.

For each protein encoding, some statistics are given on the distribution of pixel values in the sample of randomly generated proteins; issues are identified and possible solutions are given.

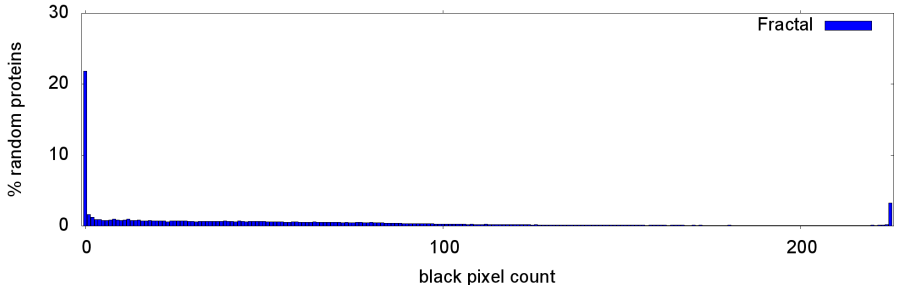
5.5.1 Fractal proteins

Studying the aggregate properties of the set of 100,000 randomly generated fractal proteins, it rapidly appears (see Figure 5.8) that fractal proteins are biased towards extremes, both in terms of pixel values and in terms of the amount of black pixels per protein. These results illustrate the above critique of fractal protein encoding in Section 5.1.

(a) Gradient of pixel values, proportional to the distribution of pixel values in the sample of fractal proteins.



(b) Distribution of the values of non-black pixels in randomly generated fractal proteins.



(c) The proportion of randomly generated fractal proteins containing given numbers of black pixels.

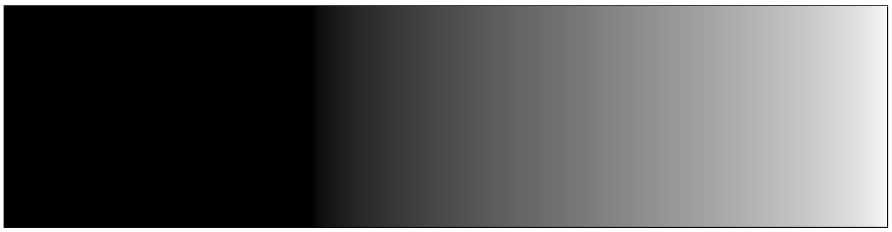
Figure 5.8: Visual statistics of randomly generated fractal proteins.

Figure 5.8a shows a gradient representing the distribution of all pixel values in the randomly generated fractal proteins; the more a shade of grey is present in proteins, the more it is present in the gradient. Consequently it is visually clear that on average a fractal protein is approximately composed for one quarter of black pixels and for three quarters of pixels close to being white; there are few pixels with intermediate values. This under-exploitation of the greyscale is confirmed by Figure 5.8b, which shows the distribution of pixels with non-black values, and is heavily biased towards white values. Note that the spike and subsequent decrease are to be expected, and are side-effects of the variations, within the space around the Mandelbrot set, of the number of iterations of the Mandelbrot set function after which a point of that space becomes unbounded and therefore not part of the set.

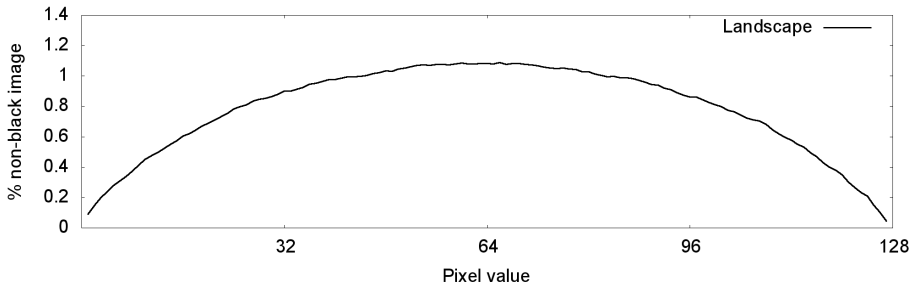
Additionally Figure 5.8c shows that about 22% of randomly generated proteins do not have one black pixel, and approximately 4% of proteins are fully black. Beyond the undesirability of fully black or white proteins (there are small variations amongst white proteins but they are functionally almost identical), the mapping of a quarter of protein definitions to one of two proteins is not desirable in a protein encoding process, as it strongly limits the expressivity of the encoding.

A first step towards a solution to these issues may be to map the Cartesian fractal coordinates $< x, y, z >$ to an ad-hoc non-Cartesian space providing more granularity to the "interesting" portions of the fractal (the borders), and effectively reducing the occurrence of uninteresting, fully black or white, proteins. However as there is at this point little reason to believe the fractal protein encoding to be superior to the landscape protein encoding, this will not be attempted.

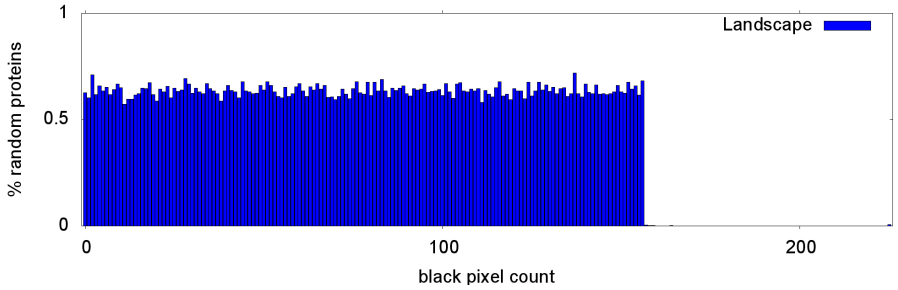
5.5.2 Landscape proteins



(a) Gradient of pixel values, proportional to the distribution of pixel values in the sample of landscape proteins.



(b) Distribution of the values of non-black pixels in randomly generated landscape proteins.



(c) The proportion of randomly generated landscape proteins containing given numbers of black pixels.

Figure 5.9: Visual statistics of randomly generated landscape proteins.

The protein statistics for landscape proteins in Figure 5.9 appear visually smoother than their equivalent for fractal proteins in Figure 5.8, reflecting a better exploitation of the greyscale and a more even distribution of pixel values used. Particularly, the smooth gradient Figure 5.9a reflects the gradients present in most landscape proteins, and contrasts heavily with the rougher transition from black to white for fractal proteins in Figure 5.8. Another illustration of the difference in evenness of distribution of pixel values is simply the difference in the vertical axis scale between the plots for fractal proteins in Figure 5.8 and those for landscape proteins in Figure 5.9.

The semi-circular shape of the distribution of non-black pixel values in Figure 5.9b is also a consequence of the gradients present in landscape proteins: middle values are more likely to be present in the gradient of two random values. The size of the black area of a randomly generated landscape protein is uniformly taken from the range $[0, 157]$, which produces proteins with 0 to 70% of black pixels ($0.7 \times 15 \times 15 \approx 157$), the resulting distribution of generated proteins with a given number of black pixels is illustrated in Figure 5.9c, where the number of proteins can be seen dropping to zero outside of the initialisation range $[0, 157]$.

One possible limitation of landscape proteins may be their simplicity, which might make them unsuitable for problems requiring more complex protein-protein interactions. A solution to this is to allow the proteins to complexify through mutation by the addition of more than the two current defining position/value pairs. This can easily be done by first adding a way-point between the two positions with the gradient value at that point before then mutating this new protein/value pair this would allow protein complexification while preserving the smooth modification of the end protein shape. The black area can be similarly split through mutation.

5.5.3 Mondrian proteins

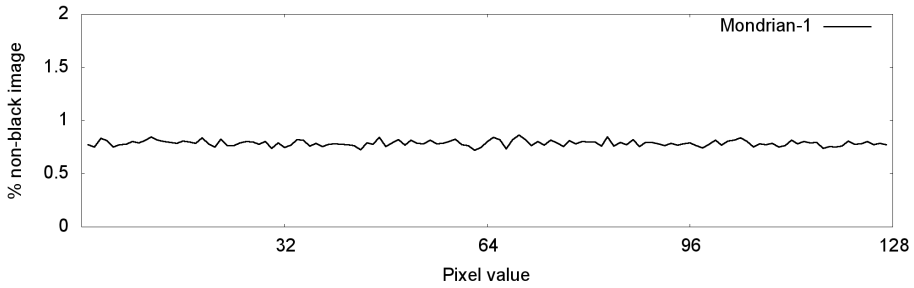
As for fractal and landscape proteins above, Figures 5.10, 5.11, and 5.12 below show statistical properties of Mondrian proteins with respectively one, two, and four Mondrian portions. Compared to fractal proteins which are composed on average of a quarter of black pixels, and landscape proteins composed on average of a third of black pixels, Mondrian proteins are composed on average of fifty percent of black pixels; such a high proportion of black pixels in the protein may be excessive. This proportion does not change with the number portions a Mondrian protein is composed of, but instead the distribution of the amount of black pixels per protein changes strongly: as the portion count increases, this distribution becomes increasingly uneven to the point where most randomly generated Mondrian-4 proteins contain either all black pixels, or no black pixels at all. This progression is obvious looking in turn at Figures 5.10c, 5.11c, and 5.12c, and is likely the reason for the poorer performances associated with Mondrian proteins with more portions. This change in distribution is due to the use of only 'minimum' and 'maximum' operations to combine Mondrian portions into a protein, which excessively favour either the black proteins (minimum), or the clearest non-black proteins (maximum). The spikes in these plots are due to the protein length cut used to generate a Mondrian portion's black part. Consequently the black pixel count in a Mondrian portion is always a multiple of 15, as is visible for the single portion protein (Mondrian-1) in Figure 5.10c.

Mondrian proteins do not contain gradients, but as opposed to fractal proteins, every non-black value is equally likely to be present in the Mondrian portions which compose a Mondrian protein. This explains the smooth gradients in Figures 5.10a, 5.11a, and 5.12a. Also, as the number of portions per protein increases, the darker values in the gradient give way to lighter values. This phenomenon is most visible in the distribution of non-black pixel values for Mondrian-4 (Figure 5.12b, and can also be explained by the use of the minimum and maximum operations to combine portions.

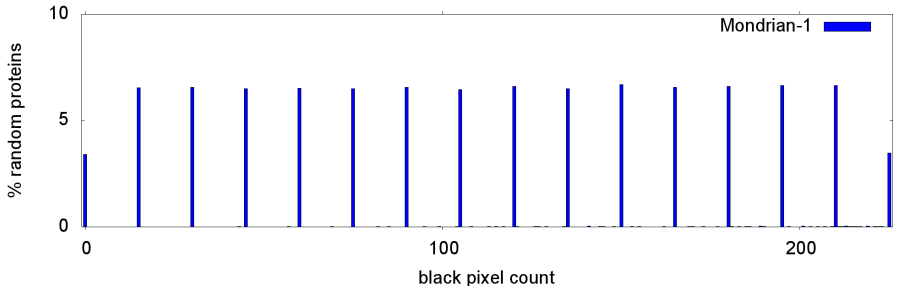
This analysis of randomly generated Mondrian proteins has revealed defaults in their design which

limit the expressivity of the protein representation. While the Mondrian-1 portion is likely too simple a protein encoding, the current operations used to combine the Mondrian portions into proteins (minimum and maximum) produce even poorer proteins. Other methods of combining Mondrian portions, careful to not emphasize too much extreme pixel values, may produce more complex yet still balanced proteins.

(a) Mondrian protein encoding with one portion (Mondrian-1); left: average protein, right: proportional gradient of pixel values.



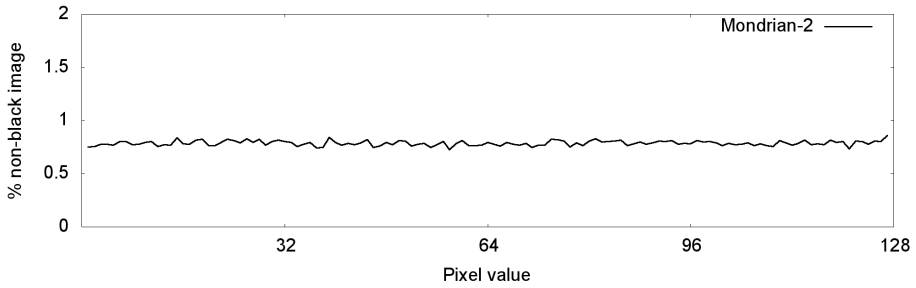
(b) Distribution of the values of non-black pixels in randomly generated one portion Mondrian proteins.



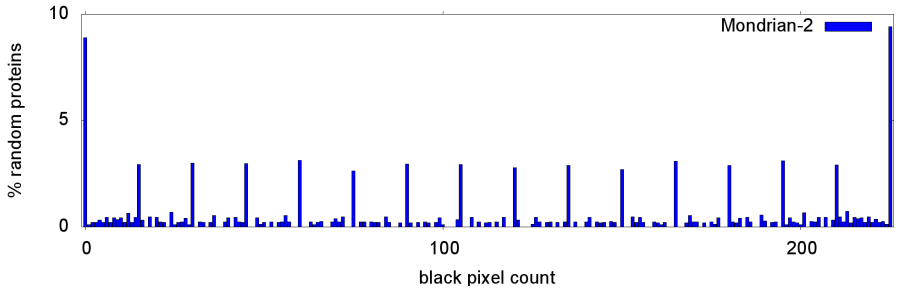
(c) The proportion of randomly generated one portion Mondrian proteins containing given numbers of black pixels.

Figure 5.10: Visual statistics of randomly generated one portion Mondrian proteins (Mondrian-1).

(a) Mondrian protein encoding with two portions (Mondrian-2); left: average protein, right: proportional gradient of pixel values.



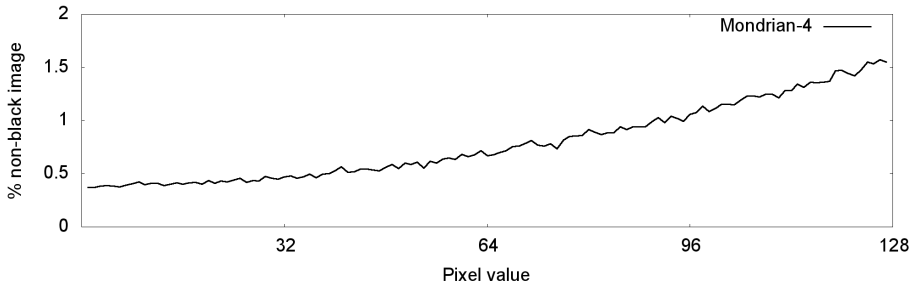
(b) Distribution of the values of non-black pixels in randomly generated two portions Mondrian proteins.



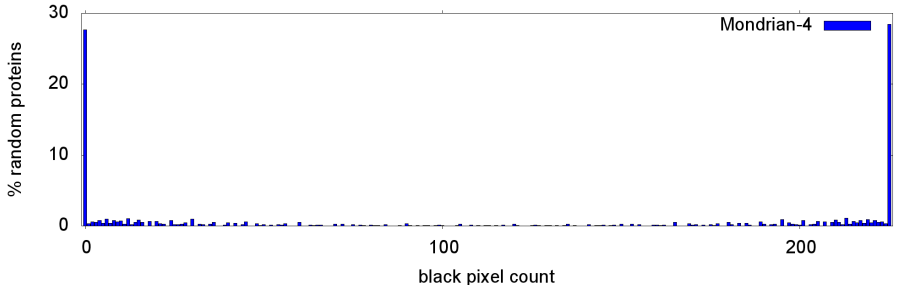
(c) The proportion of randomly generated two portions Mondrian proteins containing given numbers of black pixels.

Figure 5.11: Visual statistics of randomly generated two portions Mondrian proteins (Mondrian-2).

(a) Mondrian protein encoding with four portions (Mondrian-4); left: average protein, right: proportional gradient of pixel values.



(b) Distribution of the values of non-black pixels in randomly generated four portions Mondrian proteins.



(c) The proportion of randomly generated four portions Mondrian proteins containing given numbers of black pixels.

Figure 5.12: Visual statistics of randomly generated four portions Mondrian proteins (Mondrian-4).

5.6 Discussion

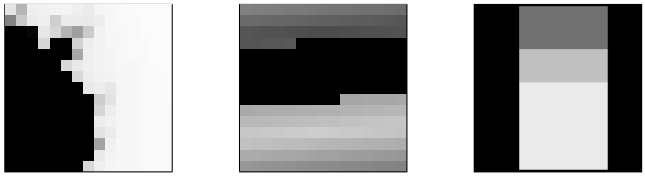


Figure 5.13: Example bitmaps from the three protein encoding methods covered in this chapter. From left to right: fractal, landscape, and Mondrian proteins. Though they are here represented as two-dimensional bitmaps, it was shown in this chapter that the two-dimensional aspect is not important in bitmap proteins.

In this chapter, limitations of the fractal protein encoding were identified, and alternative protein encodings aiming to mitigate these limitations were designed. Figure 5.13 illustrates bitmap proteins produced by each of the protein encodings studied. The alternative encodings were found to perform better or similarly to fractal encoding on both a control and developmental problems, validating the hypothesis that the strength of the FGRN model does not come from the use of fractals.

Particularly, the system using landscape protein encoding was found to perform overall significantly better ($p < 0.03$) on these problems than the one using fractal protein encoding, despite the system being optimised for the latter. A statistical analysis of large samples of randomly generated proteins for each protein encoding brought additional insights into the causes of the differences in performance between protein encodings, and showed landscape proteins to be particularly expressive. It was also shown that landscape protein encoding provides a smooth path for evolutionary complexification of the proteins.

If fractals are not the important component as was previously thought, what is? The main other feature of the FGRN model is the merging of input and regulatory proteins into a ‘cytoplasm’, which then determines the further activation of the regulatory and output genes; the addition of a protein to the cytoplasm can greatly affect this mechanism, which allows the system to radically and discretely switch its ‘regime’ while running and gives it the ability to select which regime to run in, something which is not possible using continuous models such as for instance a recurrent neural network. As the main other specificity of the FGRN model, the author believes this is the most important feature of the model, which embodies an abstraction of the workings of natural GRNs, instead of the complex protein encoding and chemistry as previously hypothesised. The next chapter will aim to extract the key mechanisms of the FGRN model and provide a model embodying these and stripping out unnecessary features. Additional goals will be to make the new model more easily extensible and more appropriate for real world use.

Chapter 6

The Input-Merge-Regulate-Output (IMRO) Architecture

The previous chapter investigated the fractal protein encoding, and identified limitations in the expressivity of the resulting fractal proteins. Alternative encodings were introduced addressing these limitations leading to improvements in the range of expressivity of the proteins produced, and computational experiments showed improved performances confirming these findings. However the resulting control system still fell short on the harder double pole balancing problem.

This chapter focuses on the other main feature of the FGRN model : the ability of an FGRN genome being 'executed' to radically change the way it behaves as a function of its current state (the combination of previously produced regulatory proteins) and the environment (the current inputs). This ability mimics that of biological cells which can exhibit a wide-variety of behaviours as a function of their environment and the regulatory proteins (transcription factors) present. Large multi-cellular organisms present the most striking example of this mechanism, each type of cell (e.g. neuron, skin, muscle) fulfilling a widely different function while still possessing the same genome.

The FGRN mechanism of merging environmental (input) proteins and regulatory proteins into a 'cytoplasm' (merged protein product) that determines further gene activation will be kept, as well as the general organisation of the genome. The surrounding details of the FGRN model — protein encoding, gene activation function, protein production and decay functions — will be simplified and wherever possible will be parameterised to make them more pliable in the evolutionary process.

In the FGRN genome, all genes have the same parameters (promoter and output protein definitions, affinity and concentration thresholds), which are used differently depending on the gene's type. This makes cumbersome any modification of the model requiring the introduction of a new evolvable parameter. However, different gene types do share some mechanisms, for instance regulatory and behavioural genes share a common activation mechanism whereas environmental and regulatory genes both produce proteins into the merged protein product (cytoplasm). The Input-Merge-Regulate-Output (IMRO) architecture, an abstraction of the FGRN model's structure, will be introduced and these shared mechanisms will be encapsulated in modules with well defined input/output interfaces. The IMRO architecture will also combine these independent modules into input (environmental), output (behavioural), and regula-

tory genes, and a merging component (cytoplasm); the receptor gene, playing a lesser part in the FGRN model, is omitted. An IMRO genome will be introduced, composed of evolvable input, output, and regulatory genes. The merging component (which takes on the same role as the FGRN model's cytoplasm) encapsulates the protein merging mechanism and in contrast to the genes has no evolvable parameter.

After presenting the IMRO architecture, an implementation of each module will be introduced, aiming for each of them to both simplify and make more evolvable each corresponding mechanism of the FGRN model. The resulting IMRO model (the combination of the IMRO architecture and of the module implementations) will then be applied to the pole balancing problems, and to a more difficult version of the acrobot swing-up problem, on which previous attempts using evolutionary methods have failed [dMSBA08].

6.1 Architecture description

6.1.1 Structure

The structure of the controller and the gene roles in the IMRO architecture are almost identical to those of a FGRN controller (see Figure 6.1). Figure 6.2 details the data flow of an IMRO controller. 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. Figure 6.3 details the decomposition of genes into sub-components.

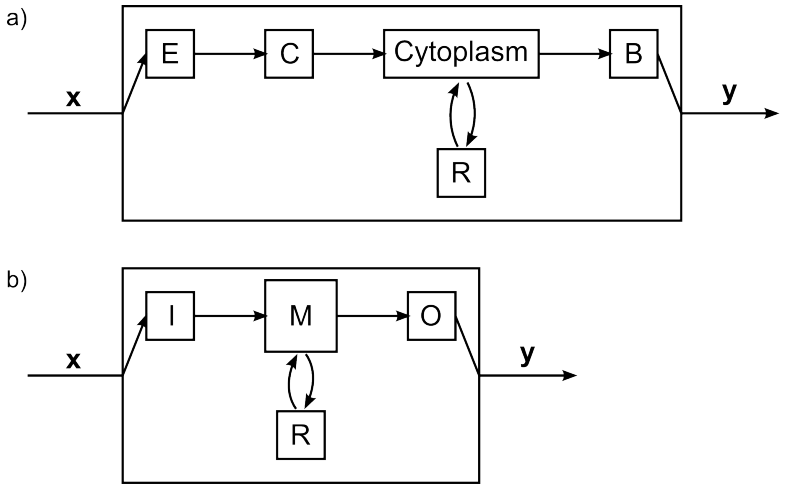


Figure 6.1: FGRN and IMRO controller structures. a) the structure of the FGRN controller, b) the structure of the IMRO controller. Aside from the different naming conventions, the only difference in the overall organisation of the controllers is the absence in the IMRO controller of an equivalent to the FGRN model's receptor (C) gene, omitted from the IMRO architecture as it plays a minimal role in the controller.

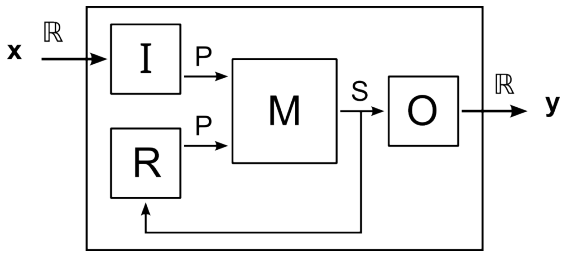


Figure 6.2: The IMRO architecture for GRN control. The data flow of the controller between the input (I), merging (M), regulatory (R), and output (O) components is also shown. Like an FGRN controller, an IMRO controller can have multiple input, regulatory, and output genes. Key: P = protein, S = cell state, $\mathbb{R}$ = real scalar

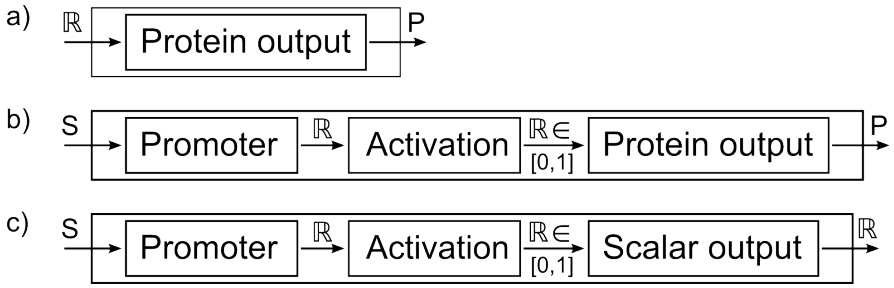


Figure 6.3: The composition of IMRO genes: a) input gene, b) regulatory gene, and c) output gene.

6.1.2 Control loop algorithm

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 "aged" (their time to live attribute is decremented by one); (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 each produce one of the controller's scalar outputs; a deactivated output gene produces a zero output. The IMRO control loop is shown in Algorithm 6.1.

Algorithm 6.1: The IMRO control loop.

```
declare array input_genes := getInputGenes(genome)
declare array output_genes := getOutputGenes(genome)
declare array regulatory_genes := getRegulatoryGenes(genome)

proteins := [ ]
while not controlEnded()
  { emit proteins for the inputs }
  declare array inputs = getInputs()
  for i := 1 in inputs.length
    append(proteins, protein_from_value(input_gene[i].protein, inputs[i])
  end

  { merge proteins into a cell state }
  declare struct state := merge(proteins)

  { age proteins, and remove expired ones }
  age(proteins)

  { activate regulatory/output genes }
  for each gene in regulatory_genes, output_genes
    matching_score := match(gene.promoter, state)
    activation_value := activate(matching_score)

    gene.activated := false
    if activation_value > 0
      gene.activated := true
      if gene.is_regulatory
        append(proteins, protein_output(gene, activation_value))
      end
      if gene.is_output
        gene.output = scalar_output(gene, activation_value)
      end
    end
  end

  { extract outputs }
  declare array outputs := [ ]
  for each gene in output_genes
    append(outputs, gene.output)
  end
  setOutputs(outputs)
end
```

6.2 Proteins and cell state merging

The protein bitmaps in the FGRN model are replaced by smaller vectors of fixed size, which is also the size of the promoter mask vectors and weight vectors, and of the protein output value vectors and mask vectors. The merging of output proteins in IMRO is identical to the FGRN model's equivalent merging of protein bitmaps into the cytoplasm.

In detail, 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 of the proteins with the maximum level, if several proteins have the same maximum level); 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. This merging process is illustrated in Figure 6.4.

In the initial population of genomes, the protein levels are set to random integer values in the range $[-L_{max}, L_{max}]$, where L_{max} is constant across genomes. Consequently, each regulatory protein in the genomes of the initial population affects on average half of the cell state, the portions of the protein with negative level values being ignored by the running system.

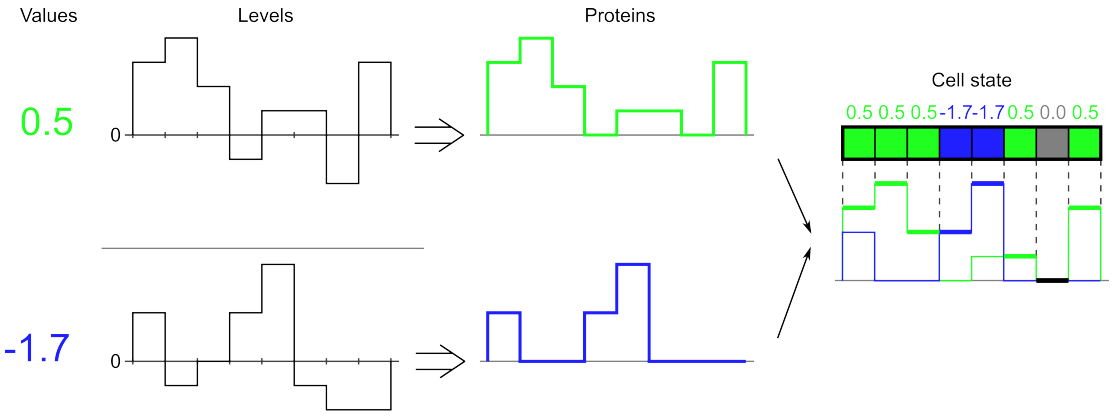


Figure 6.4: The merging of two proteins in the IMRO model. On the left the level arrays which form the protein definitions are illustrated, the coloured number representing the real value currently associated with the protein; in the middle, the resulting proteins are shown; on the right the proteins are merged at every level position, producing the coloured cell state array on top.

6.3 Gene components

As detailed in Figure 6.3, the genes are composed of combinations of four components: promoter, activation, protein-output, and scalar-output. All component parameters are subject to evolution.

6.3.1 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.

In the initial, randomly generated, population of genomes, the weights are taken uniformly from the range $[-W_{max}, W_{max}]$, W_{max} being a constant; each value in M is initialised uniformly from the boolean set $0, 1$. Evolutionarily, each component of W is treated as a separate parameter whereas M is treated as a single parameter. When M is mutated, one of its component is randomly selected to be inverted.

6.3.2 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

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

This allows for 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 [BGH03]. 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).

In the initial population of genomes, the threshold θ is uniformly taken from the range $[-1, 1]$ and kept within these bounds; the mutation range is set to half of the initial range, and the resulting mutated values are clipped back if necessary to stay within $[-1, 1]$. The scale α is initially taken uniformly from the range $[-\alpha_{max}, \alpha_{max}]$.

6.3.3 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 the initial genome population, the scale β is taken uniformly from the range $[-\beta_{max}, \beta_{max}]$, and τ is taken from the range $[0, \tau_{max}]$, where τ_{max} is a constant; in keeping with rest of parameters, the mutation range is half of the initialisation range, in this case $[-\tau_{max}/4, \tau_{max}/4]$. In input genes, the protein output component emits a protein at every time step; to avoid a flooding of the cell state, these are permanently set with a lifespan τ of one, and their levels are initialised to 0, except for one level (different for each input) which is initialised to L_{max}.

6.3.4 Scalar output

The scalar output component determines the return value of an output gene. Depending on the problem, a boolean or an output value may be desired; in either case, the component relies on a single parameter $\gamma \in [-1, 1]$. For output gene i at time t, the boolean output $o_{i,t}$ is given by

$$o_{i,t} = \begin{cases} a_{i,t} \geq \gamma_i & \text{if } \gamma_i \geq 0 \\ a_{i,t} \leq -\gamma_i & \text{if } \gamma_i < 0 \end{cases}$$

whereas alternatively the real output value $r_{i,t} \in [0, 1]$ is given by

$$r_{i,t} = \begin{cases} \max(a_{i,t} - \gamma_i, 0) & \text{if } \gamma_i \geq 0 \\ -\min((a_{i,t}, -\gamma_i) + \gamma_i) & \text{if } \gamma_i < 0 \end{cases}$$

In the case of bang-zero-bang control, as used in the acrobot problem below, three output genes with real value output are used, each corresponding to one of the possible control signals. The control signal for which the corresponding gene outputs the highest value is sent. In the initial genome population, γ is uniformly taken from the range $[-1, 1]$. It was found in preliminary experiments that the system is particularly sensitive to this parameter's mutation, so a lower mutation range of $[-0.1, 0.1]$ is used here.

6.4 Experiments

In this section some relevant parameter settings and experimental details for activation pattern generation, pole balancing, and the acrobot are first given. The results of these experiments are then detailed.

A maximum of 10,000 genomes are evaluated per run for the pattern generation and pole balancing experiments. For the acrobot, following the work of da Motta Salles Barreto and Anderson [dMSBA08], a maximum of 3,000 genomes is evaluated per run. All experiments are run 50 times. [1]

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 pattern generation experiments in which FGRN performances were poorer with fewer than four regulatory genes.

Unless previously specified otherwise, the mutation range of each parameter is half that of the initialisation range, e.g. for the initialisation range $[-X, X]$, the mutation range is $[-X/2, X/2]$. Mutation values are taken uniformly from the mutation range.

The size N of the cell state is set to 8 to both accommodate the number of inputs for all the problems studied here, and provide some space for regulatory interactions. The maximum level value L_{max} of a protein is set to 128 (which is the maximum value of a protein in the FGRN system). The maximum initial lifespan τ_{max} of a regulatory protein is set to 4; the right value for this parameter is very much problem dependent, the value 4 being a good compromise here between the immediate reactions needed for control problems, and the longer time scale of some pattern problems. The maximum initial values for the multiplicative parameters W_{max}, α_{max}, and β_{max} are set to 8, which proved adequate in preliminary experiments.

Note that except for the size of the cell state, evolution can bring the genome values of these parameters outside of these bounds. These constants entirely define both the behaviour of the IMRO system and the evolutionary behaviour of its parameters.

FGRN

The FGRN genomes evolved use the original fractal protein encoding, and are composed of four regulatory genes, one receptor gene, one behavioural (output) gene, and as many environmental genes as there are inputs. The zero centred input-mapping scheme, with negative protein concentrations introduced in Section 4.4 is used here. All other FGRN parameters are set as in the previous chapters.

[1] The source code for all the experiments and systems described in Chapters 4, 5, and 6 is available at http://github.com/susano/ppsn2012

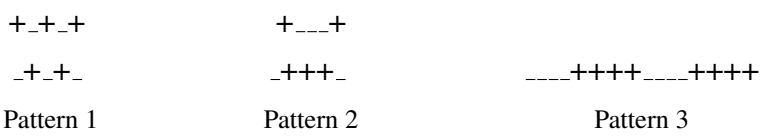


Figure 6.5: Test activation patterns from [Ben04b]. Patterns 1 and 2 require two separate output genes per genome.

Genetic algorithm

IMRO and FGRN genomes are evolved using the ALPS genetic algorithm [Hor06] with a layer size of 25, an age gap of 10, and the polynomial ageing scheme. 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 mutation rate is 0.1, and uniform crossover is always applied. In Section 4.3, ALPS was found to increase the reliability with which successful FGRN controllers were found. In the FGRN experiments throughout this thesis, successful solutions rarely had a genome composition different from the initial population's genomes; it is therefore likely that these mutations are disproportionally deleterious. For these experiments, genome level mutations will not be used.

Control problems

The setup of the pole balancing and acrobot problems is detailed in Sections 2.1.2 and 2.1.3. The controllers are run on the pole balancing problems for 100,000 simulated timesteps ($\approx$ 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 in da Motta Salles Barreto and Anderson's work [dMSBA08], to allow the genetic algorithm to find initial solutions from which to start improving. Though this effectively changes the problem, it does not directly affect the quality of the final solutions found.

Note that the acrobot problem's objective is opposite that of the pole balancing problems: whereas in the latter the system aims to stabilise the system, by maintaining the poles upright, the acrobot is an underactuated system that must be perturbed out of its stability zone.

Activation pattern generation

The initial test [Ben04b] of the FGRN model's developmental capabilities was to attempt to evolve genomes able to produce specific activation patterns (see Figure 6.5), no input was given. The fitness of a genome was the number of matches between its activation output and the pattern.

While the focus of the IMRO system is control, 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 therefore both FGRN and IMRO genomes were applied to this task.

Table 6.1: The percentage of successfully generated patterns, and the mean number of evaluations required to success (standard deviation in parenthesis).

	FGRN		IMRO	
Pattern 1	100%	2434(2271)	100%	225(255)
Pattern 2	100%	1073(1102)	100%	160(119)
Pattern 3	68%	9554(4986)	100%	1168(1142)

6.4.1 Results

Activation pattern generation

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 [Ben04b], where an additional guidance component needed to be added to the fitness to successfully evolve this pattern. This can be attributed to the use here of the ALPS genetic algorithm, and to an improvement in the FGRN settings used.

Pole balancing

The results are detailed in Table 6.2. 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 being only able to balance the two poles for 228 timesteps ($\approx$ 5 seconds) out of the required 100,000. Figure 6.6 and 6.7 illustrate the performances of respectively the IMRO and FGRN systems learning to solve the single pole balancing problem. Figure 6.8 illustrates the learning of the IMRO system solving the double pole balancing problem.

Table 6.2: Number of failures/evaluations before a successful controller is found. Note that the earliest successful solutions for the single pole balancing were found in the first and second generations. Key: SD = Standard Deviation

	FGRN			IMRO		
	Mean(SD)	Best	Worst	Mean(SD)	Best	Worst
Single Pole	306(303)	35	2005	156(111)	5	589
Double Pole	-	-	-	1677(1261)	245	5719

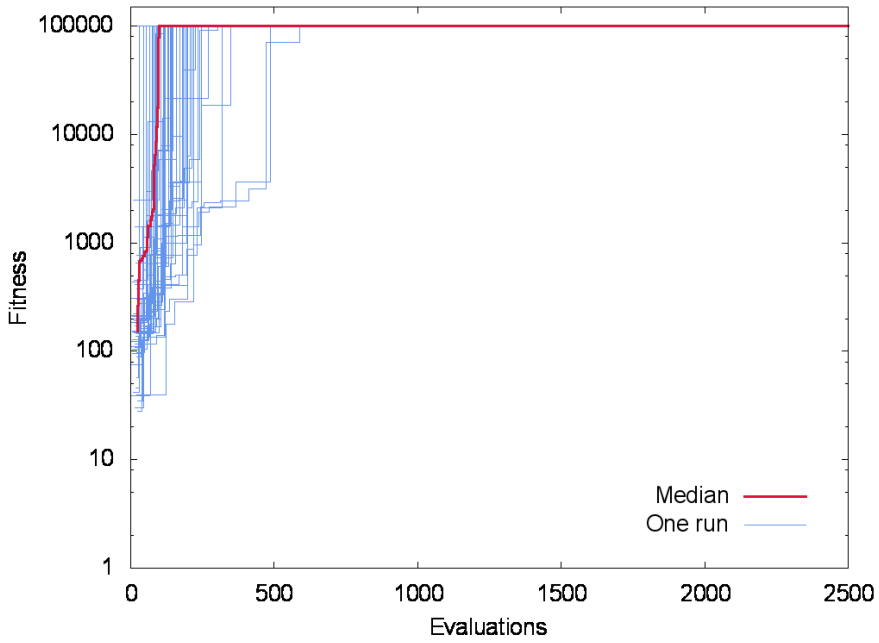


Figure 6.6: The IMRO system learning on the single pole balancing problem (SPB). Each blue line represents one of the fifty runs, and the bold red line is the median of these runs at each point.

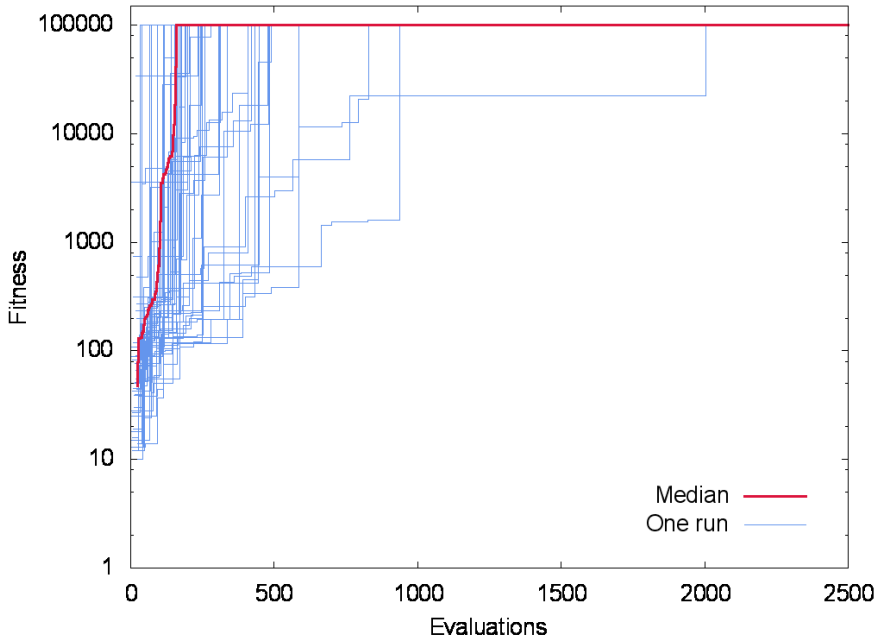


Figure 6.7: FGRN system learning on the single pole balancing problem (SPB)

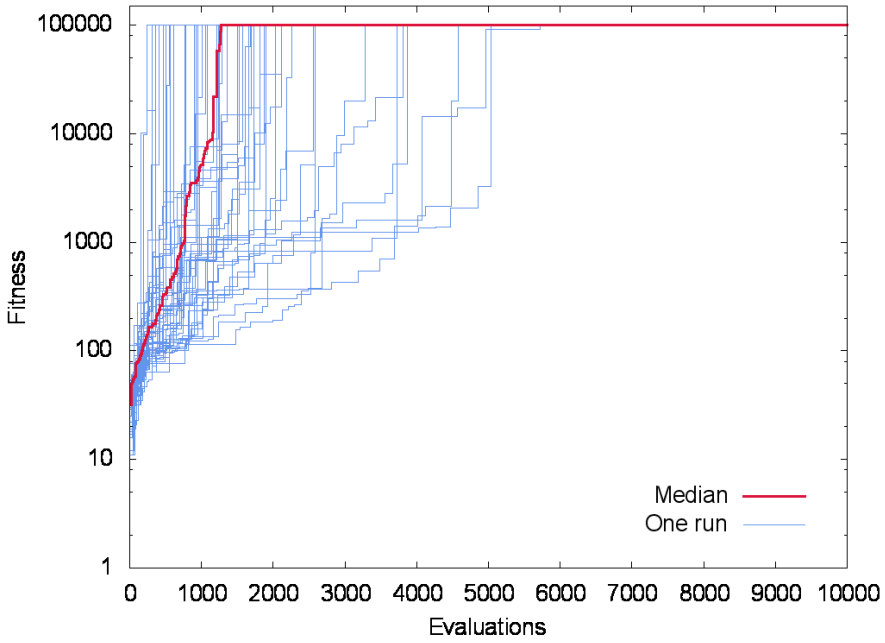


Figure 6.8: IMRO system learning on the double pole balancing problem (DPB)

Table 6.3: 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 [dMSBA08].

	Mean(SD)	Best	Worst
SARSA-RGD	276.6(106.6)	238	-
IMRO	304.3(41.2)	255	497
LSPI	335.9(12.1)	315	343
FGRN	435.7(172.1)	270	1050

Acrobot

Table 6.3 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 6.9 and 6.10 illustrate the performances of respectively the IMRO and FGRN systems in solving the acrobot problem. Figure 6.11 shows the trajectory of an acrobot controlled by the IMRO system.

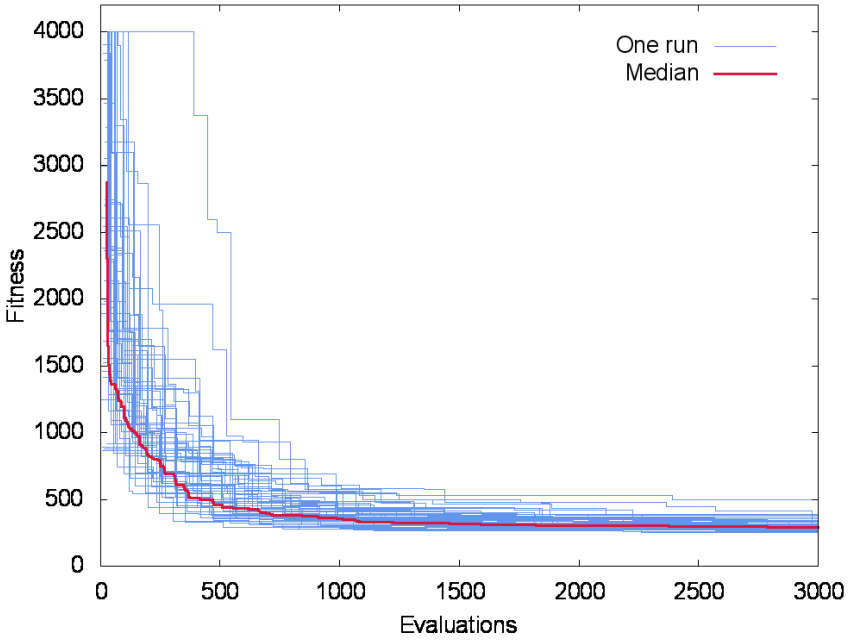


Figure 6.9: IMRO system learning on the acrobot problem

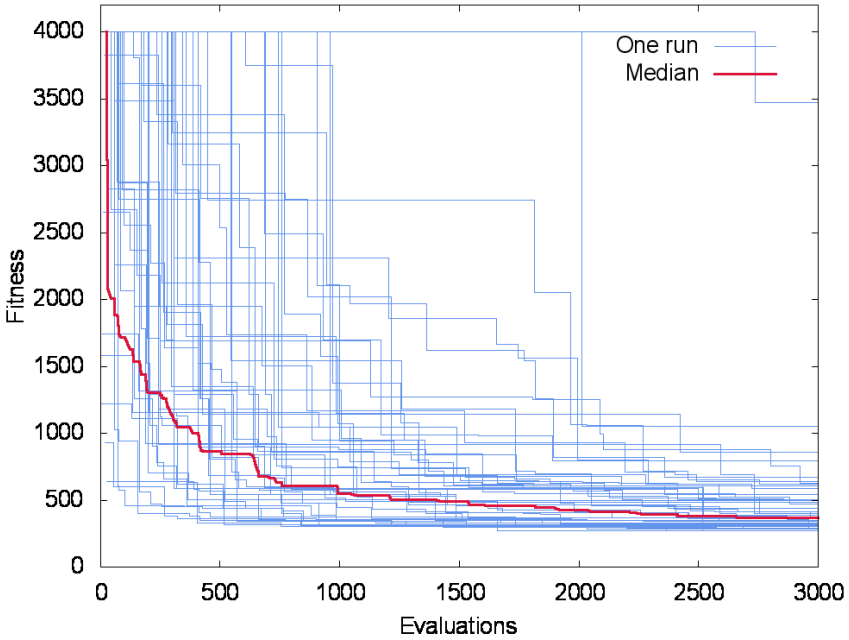


Figure 6.10: FGRN system learning on the acrobot problem

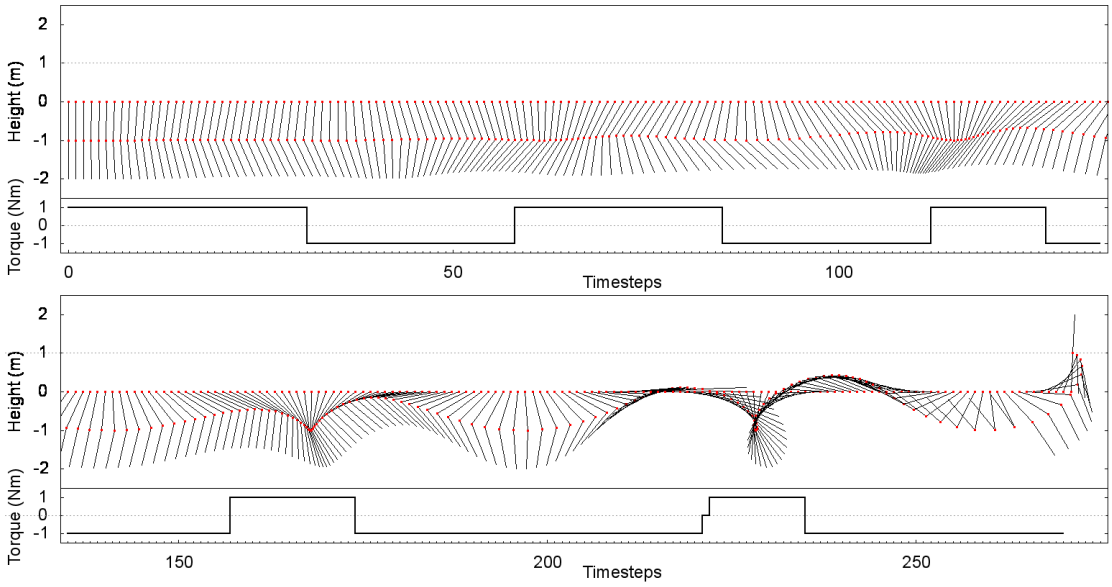


Figure 6.11: An example acrobot swing-up trajectory produced by the IMRO system. Top, the position of the acrobot at each time step. Bottom, the force applied at each timestep. Long periods of the same activation, and limited use of the null force action, are typical of efficient swing-up solutions.

6.5 Discussion

In this chapter, the IMRO architecture was introduced, which focuses on the regime-switching feature of biological GRNs which was also a part of the FGRN model. All other components of the FGRN model were simplified and made more evolvable where possible. These simplifications 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 IMRO model was also shown to be successful on the pattern generation problems.

The performance of the FGRN system in the same experiments, and particularly its combined failure on the double pole balancing problem and in the generation of pattern 3, and its limited success on the acrobot, hints that it would be inadequate for use on harder control problems. The introduction of the IMRO architecture also provides further practical advantages over the FGRN model:

Ease of extension

Modularity. The monolithic nature of the FGRN model which depending on a gene's type uses the same gene data structure in different ways, makes modification/extension of the model difficult. In the IMRO architecture, a modular, type-dependent, gene structure was introduced instead, conserving similar gene roles. The modules internal to these gene structures, with clearly defined interfaces, were reused across gene types (e.g. regulatory and output genes still share the same activation mechanism). This allows for the independent, simultaneous, modification of the model's sub-mechanisms (protein representation/chemistry, gene activation, system input/output).

Removal of arbitrary constants. The FGRN model also contains multiple arbitrary constants in its

mechanisms for activation, protein production, and protein decay. In the IMRO architecture, these mechanisms were redesigned, preserving their functionality, but removing the need for arbitrary constants and wherever possible making these mechanisms subject to evolution.

Real world ease of use

The original FGRN model limited inputs to the $[0, 1]$ range, and even with the introduction in Chapter 4 of negative concentrations extending the range of possible inputs to $[-1, 1]$, having a bounded range of possible input values as opposed to the full range of real values limits the usefulness of the system on real world problems. The inputs would therefore need to be normalised to the acceptable range, which might require, depending on the problem, the use of historical data; the robustness of the system when faced with extra-ordinary inputs would then be limited by the need to 'clip' the input to fit in the range. In the IMRO architecture, The need for a fixed input range and the associated problems were removed. While the ability of the FGRN model to provide boolean (based on gene activation), and real (based on protein concentration) output was preserved.

The complete superiority of the IMRO system's performance over that of the FGRN system is encouraging. The next chapter will focus on evaluating the applicability of the IMRO system on multiple, yet untested, aspects of control; and an attempt will also be made to add to the system the best performing generative protein encoding from Chapter 5, landscape protein encoding.

Chapter 7

IMRO Applicability

The previous chapter introduced the Input-Merge-Regulate-Output (IMRO) architecture and presented a simple model implementing it. The resulting system was found to perform significantly better than the FGRN system on both developmental (pattern generation) and control (the single pole balancing and acrobot) problems. The system was also able to solve the double pole balancing problem, for which the FGRN system failed to produce any successful controller.

However, there are still some features displayed by the FGRN model and essential features of a genetic reinforcement learning control system which the IMRO model has not yet displayed. To fill these gaps, in this chapter the IMRO model will be showcased on the following issues:

- **Generative protein encoding.** As opposed to the FGRN system, the simple IMRO system introduced in the previous chapter does not contain a generative protein encoding step. Chapter 5 introduced landscape protein encoding, a generative protein encoding scheme which performed significantly better than fractal protein encoding on both control and developmental problems. Bentley found the addition of a generative protein encoding step to the FGRN system increased the evolvability of the system [Ben05]. In this chapter, the IMRO system will be combined with the landscape protein encoding in the hope for similar improvements.

- **Memory.** The ability to generate successful controllers with inputs only partially describing the state of the system controlled (e.g. the pole balancing without velocity inputs) is an important feature of the FGRN system. Although the pattern generation developmental tasks required the IMRO system to keep internal awareness of its position during the task, none of the control tasks in the previous chapter required it to keep in memory some trace of previous external inputs. To verify that the IMRO system preserves the FGRN system's ability to produce successful controllers in these conditions, it will be tested on the single and double pole balancing problems without velocity inputs.

- **Real valued outputs.** Many real-world control applications require real valued outputs, as opposed to more constrained output types such as bang-bang and bang-zero-bang, which have been used so far. The IMRO system had real valued outputs for the bang-zero-bang control of the acrobot, the highest of three outputs determining which of the possible overall control system output

$\{1, -1, 0\}$ was produced at any given timestep; however this only required one output to be greater than the others, and did not necessitate the precision that can be required of a direct real valued control signal.

The ability of the IMRO system to produce controllers outputting a real valued control signal instead of bang-bang or bang-zero-bang will be tested on the single and double pole balancing problems, with or without velocity inputs.

- **Generalisation.** This work has focused so far on the speed and reliability of learning: the ability of the genetic reinforcement learning system to learn a successful control strategy as quickly as possible, from a set starting point. Another aspect of genetic reinforcement learning focuses on the ability of the system to learn a more general version of the problem; this is generally done by varying the initial conditions (e.g. for pole balancing, the position of the cart on the track, and the angle of the pole), while keeping the problem's dynamics identical.

  A standard generalisation test exists for the single pole balancing problem [WDDA93], and has been used as a test for different versions of another GRN model [NSB10][MNH+12]. The generalisation test will be applied to IMRO controllers.

- **Problem variety.** The IMRO system was applied in the last chapter to both developmental and control problems, but a domain of applicability as wide as possible is desirable, and for completeness the IMRO system will also be applied here to the classical mountain car problem.

In this chapter, first the combination of the IMRO model introduced in Chapter 6 and of the best performing protein encoding introduced in Chapter 5 will be presented. The resulting system will be applied side-by-side with the original IMRO system to variations of the pole-balancing problem illustrating the ability of both systems to keep some knowledge of previous states to improve control. The addition of landscape proteins to the IMRO system will not be found to bring the improvements hoped for, performing similarly to the original IMRO system on most problems and less reliably on the double pole balancing. The original IMRO system will then be successfully applied to the real output versions of the pole balancing, a version of the pole balancing aiming to evaluate the system's ability to generalise, and another classical control problem, the mountain car problem.

7.1 Generative encoding and memory

The IMRO system presented in Chapter 6 uses a *direct* protein encoding, with a one-to-one mapping between the components of the protein definition in the genome to the corresponding portions of the resulting protein used in the system. In contrast, generative protein encodings such as the ones described in Chapter 5 (including the original fractal protein encoding) are *indirect*, as each component of a protein definition in the genome guides a generative process, the outcome of which is the protein corresponding to that definition. Consequently, the mutation of a single component of the protein definition influences multiple (sometimes all) aspects of the result protein. This allows the use of bigger and more complex proteins without needing to expand the search space of genomes, as the protein definitions in the genomes are kept small compared to the end protein. For the FGRN system, Bentley found the addition of the generative step to increase the system's performance on some developmental problems [Ben05].

Landscape protein encoding was the most successful protein encoding in Chapter 5, with which the FGRN system performed best on a series of developmental and control problems, outperforming Bentley's original fractal protein encoding. Landscape protein encoding will be here integrated in the IMRO system, and the resulting system will be tested on both developmental problems and control problems and its results compared to those of the IMRO system without this generative protein encoding step.

Adapting the IMRO system to use landscape proteins

The aim being to test specifically the effects of the addition of generative landscape protein encoding to the IMRO system, the modular design of IMRO is helpful here by allowing to change the protein output component while keeping mostly identical the rest of the system's component.

The IMRO protein output is modified so that the level array is replaced by a landscape protein. The settings and evolutionary characteristics of the landscape proteins are identical to those used in Chapter 5, with the small exception that the width of the protein is reduced by one from 225 to 224 ($= 8 \times 28$), so that the size of a protein is a multiple of the number of state required by the IMRO gene promoters. The landscape proteins are wider than the IMRO protein level array, but are merged using exactly the same algorithm; the resulting state is then split in 8 equal portions of length 28, the mean of each portion being taken to form the state presented to the promoters. The abbreviation 'IMRO(Landscape)' is used below to designate the resulting system.

Experiments

The IMRO(Landscape) system is run on all experiments from the previous chapter: the developmental pattern generation problems, and the pole balancing and acrobot problems. The experimental setup for all experiments is also identical to that of Chapter 6. The original IMRO results are also shown for comparison.

Additionally, it should be noted that for the control problems to which the IMRO system has been applied so far, the full state of the controlled system was given in the inputs at every time step. To assess the ability of the IMRO system to both keep an internal state 'memorising' the current situation of the controller and to act based on this state, the performance of both versions (with and without

landscape encoding) of the system will additionally be evaluated on the single and double pole balancing problem without velocity inputs. The removal of these inputs renders incomplete the state presented to the controller at each time step, forcing in optimal controllers the evolution of mechanisms to internally store information about previous states. For these additional experiments, the inputs and scaling factors used are identical to those described in Chapter 4, in Tables 4.2 and 4.4.

On pattern generation experiments, there is no significant difference in performance one way or the other between the original IMRO system and the IMRO(Landscape) system (See Table 7.1). The results on the pole balancing problems (See Table 7.2) are more interesting; though there is little difference in their performance on the single pole balancing problem (with or without velocity inputs), IMRO(Landscape) performs significantly ($p < 0.001$) worse on the double pole balancing problem, and in some runs could not provide a successful controller at all within the number of evaluations imparted.

On the single pole balancing problem without velocities, both versions of IMRO reliably find successful controllers with similar performances, but disappointingly neither version of IMRO was able to produce a successful controller on the double pole balancing problem without velocity inputs.

As for the double pole balancing, the results of IMRO(Landscape) on the acrobot problem (See Table 7.3) are significantly ($p < 0.001$) worse than that of the original IMRO, yet still significantly ($p < 0.001$) better than the FGRN system's.

Table 7.1: Results of the IMRO(Landscape) system on the pattern generation experiments: the percentage of successfully generated patterns, and the mean number of evaluations required until success (standard deviation in parenthesis).

	IMRO		IMRO(Landscape)	
Pattern 1	100%	225(255)	100%	143(109)
Pattern 2	100%	160(119)	100%	216(186)
Pattern 3	100%	1168(1142)	100%	1102(958)

Table 7.2: Results of the IMRO(Landscape) system on pole balancing problems: number of controller evaluations before a successful controller is found. Key: SD = Standard Deviation, NV = No Velocities, Succ. = success.

	IMRO				IMRO(landscape)			
	Succ.	Mean(SD)	Best	Worst	% Succ.	Mean(SD)	Best	Worst
Single Pole	100%	156(111)	5	589	100%	143(121)	4	497
Double Pole	100%	1677(1261)	245	5719	92%	3324(1674)	237	7438
Single Pole(NV)	100%	2283(1433)	63	5840	100%	2241(1736)	241	7756
Double Pole(NV)	0%	-	-	-	0%	-	-	-

Additionally, for each control problem a figure detailing the evolutionary behaviour of IMRO(Landscape) is given on top in the following pages, with the corresponding figure for the original IMRO system underneath. Figures 7.1 and 7.2 cover the single pole balancing; Figures 7.3 and 7.4, the double pole balancing; Figures 7.5 and 7.6, the single pole balancing without velocity inputs; Figures 7.7 and 7.8, the double pole balancing without velocity inputs; and Figures 7.9 and 7.10, the acrobot problem.

Conclusion

Both original and landscape versions of the IMRO systems were able to reliably find successful controllers on a control problem which required to keep track of the control system's state (the single pole balancing without velocity inputs). However neither was able to generate a successful solution for the velocity-less version of the harder double pole balancing problem. Given that successful controllers were found for the full state version of the problem, it seems likely that changes allowing the system to form more complex cell states from the proteins present would be necessary and may be sufficient for an IMRO system to be able to generate successful controllers for the double pole balancing problem without velocity input. An alternative may be to allow the complexification to occur in the promoters rather than the cell state, which would have similar effects.

The IMRO(Landscape) system's performance was not significantly distinguishable from the original IMRO system's on most problems. But on the harder double pole balancing and acrobot problems the IMRO(Landscape) system was found to perform significantly less well than the original IMRO system. Consequently there is little reason to keep the more complex Landscape protein encoding, and the remaining work in this chapter will be based on the simpler original IMRO system.

Table 7.3: Results of the IMRO(Landscape) system on the acrobot problem: the length of the shortest trajectory found to reach acrobot swing-up. The results for the FGRN and IMRO systems are taken from the previous chapter. Key: SD = Standard Deviation.

	Mean(SD)	Best	Worst
IMRO	304.3(41.2)	255	497
IMRO(Landscape)	335.8(43.6)	272	442
FGRN	435.7(172.1)	270	1050

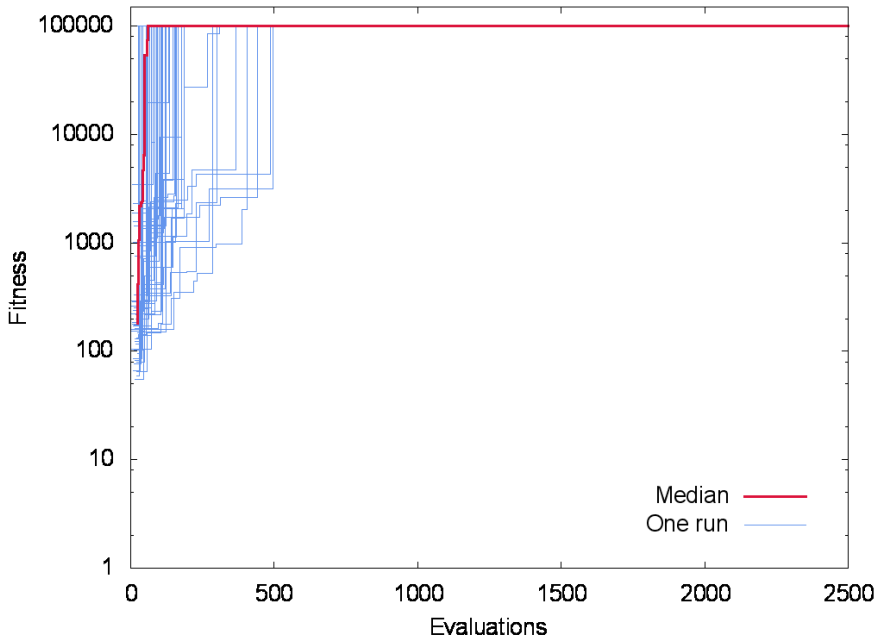


Figure 7.1: The IMRO(Landscape) system learning on the single pole balancing problem (SPB). Each blue line represents one of the fifty runs, and the bold red line is the median of these runs at each point.

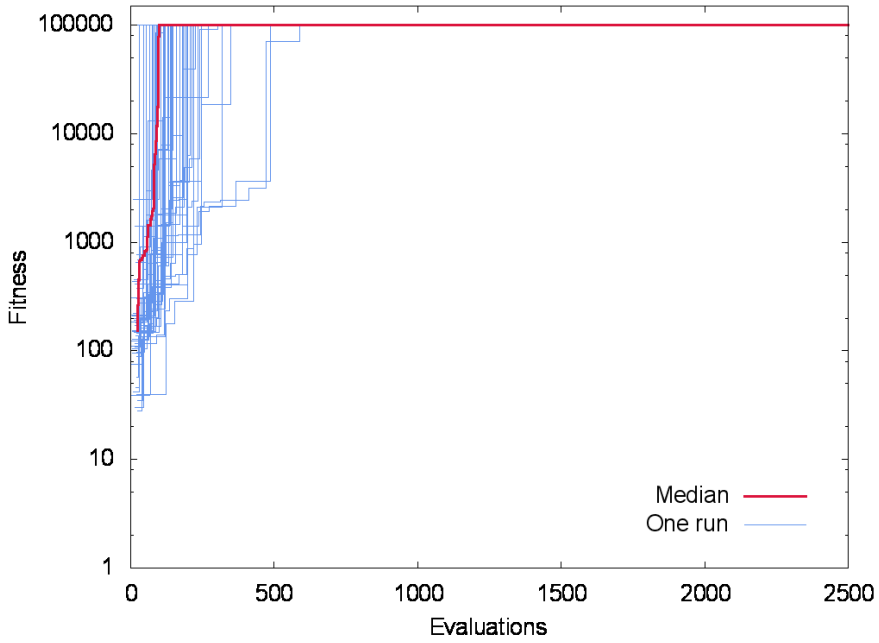


Figure 7.2: The IMRO system learning on the single pole balancing problem (SPB).

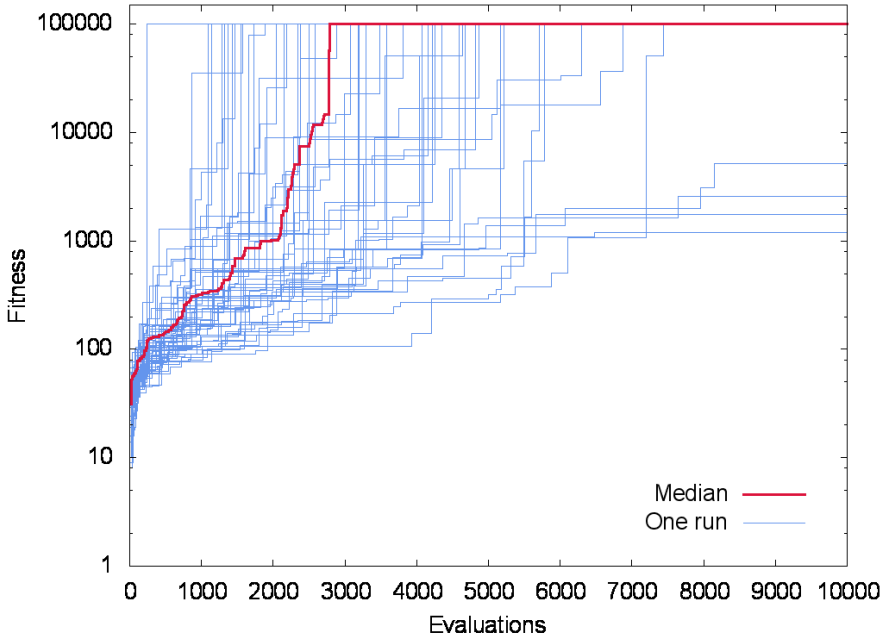


Figure 7.3: The IMRO(Landscape) system learning on the double pole balancing problem (DPB).

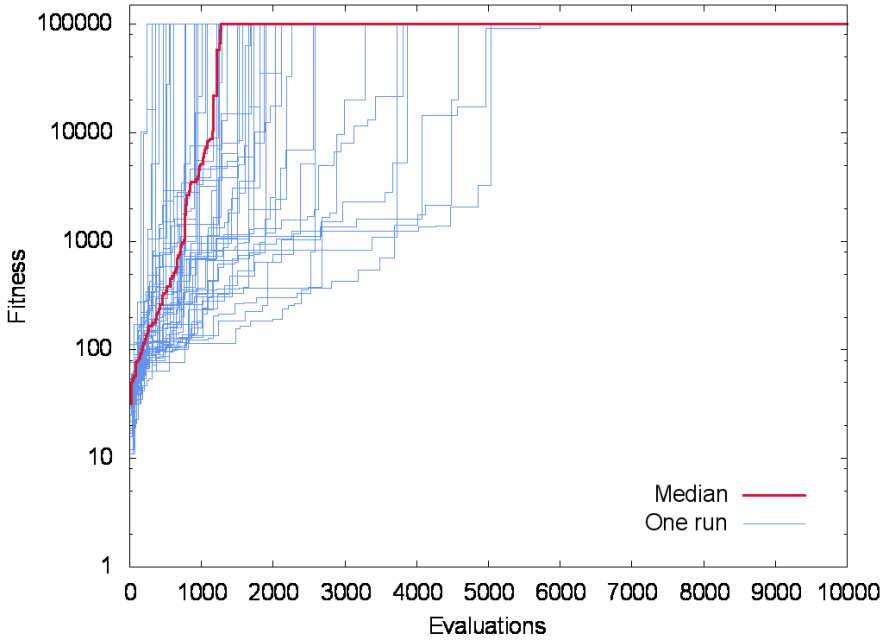


Figure 7.4: The IMRO system learning on the double pole balancing problem (DPB).

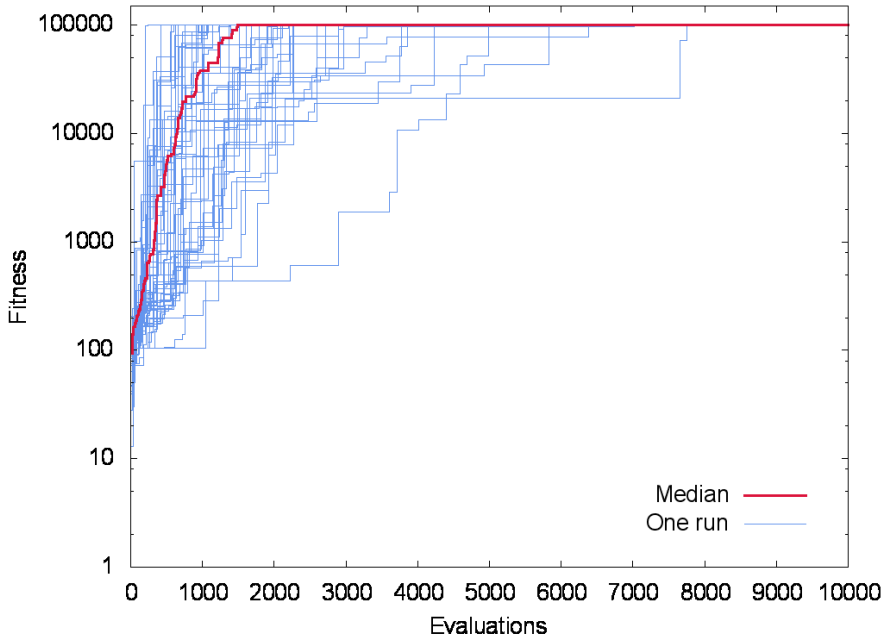


Figure 7.5: The IMRO(Landscape) system learning on the single pole balancing problem without velocity inputs (SPB(NV)).

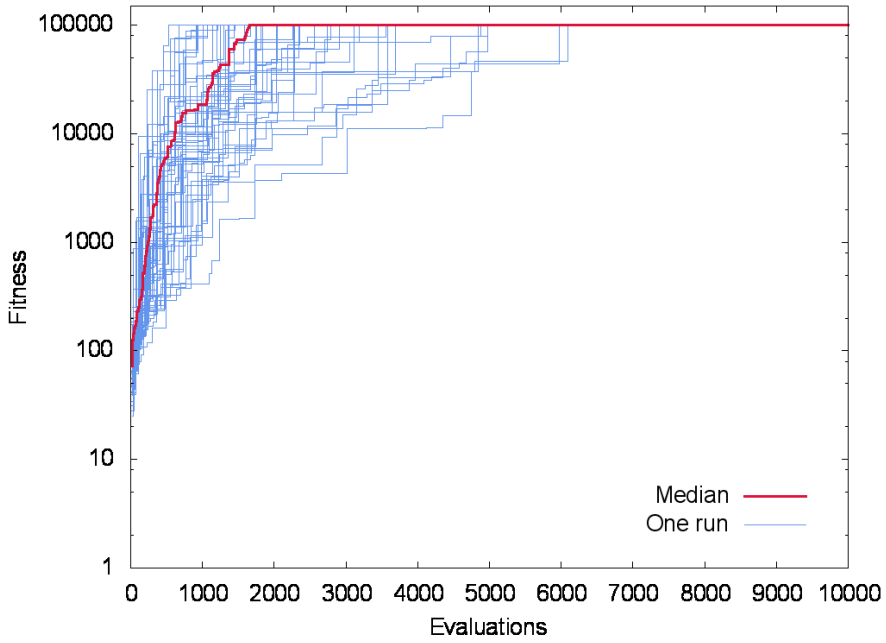


Figure 7.6: The IMRO system learning on the single pole balancing problem without velocity inputs (SPB(NV)).

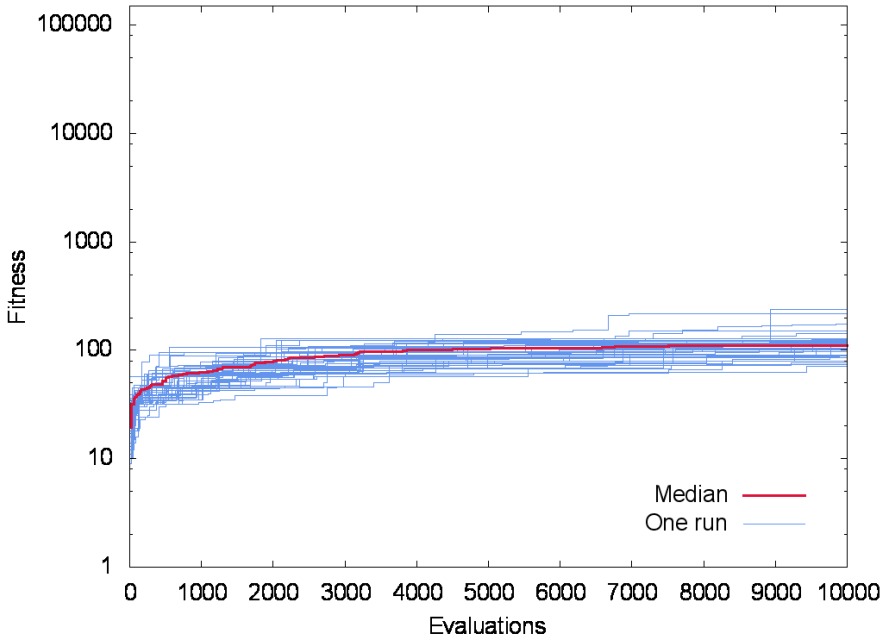


Figure 7.7: The IMRO(Landscape) system learning unsuccessfully on the double pole balancing problem without velocity inputs (DPB(NV)).

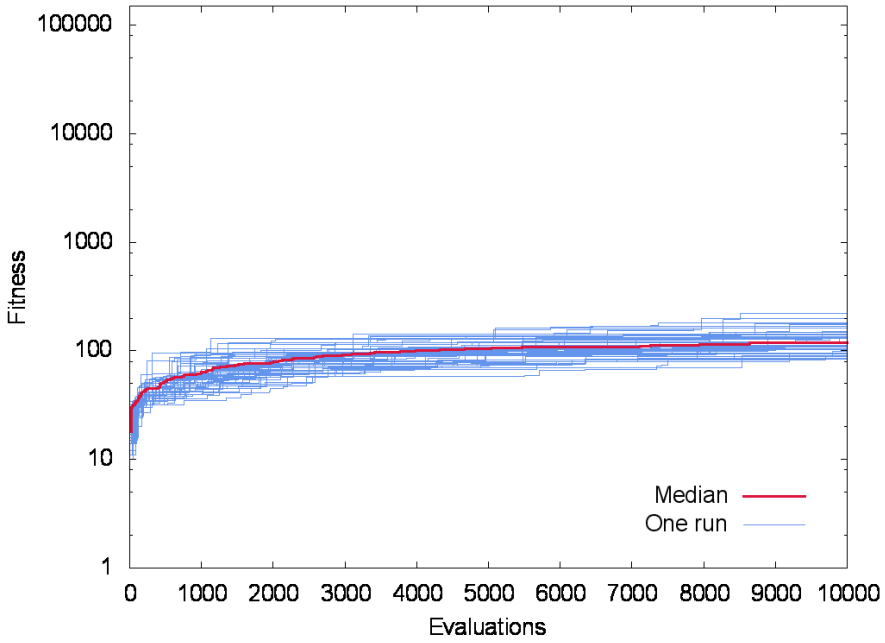


Figure 7.8: The IMRO system learning unsuccessfully on the double pole balancing problem without velocity inputs (DPB(NV)).

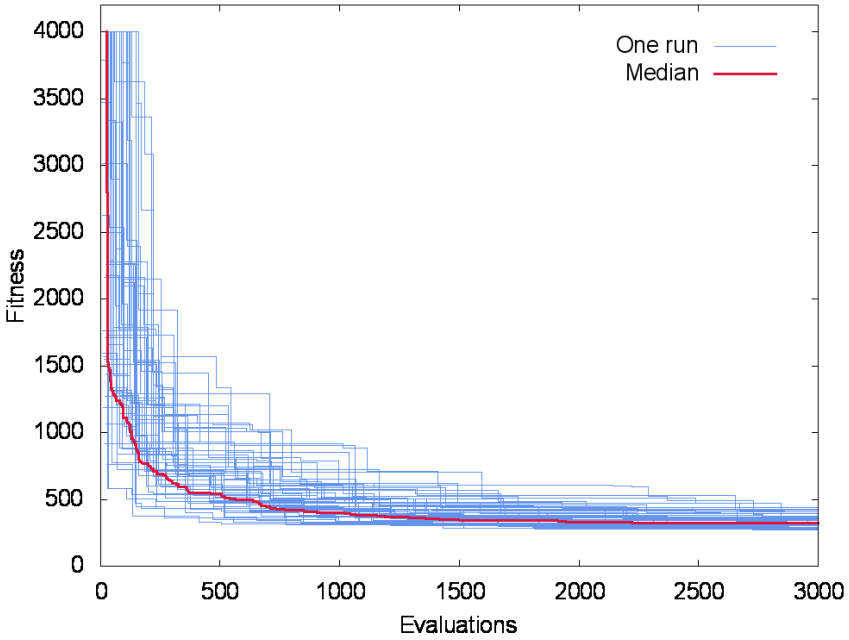


Figure 7.9: The IMRO(Landscape) system learning on the acrobot problem.

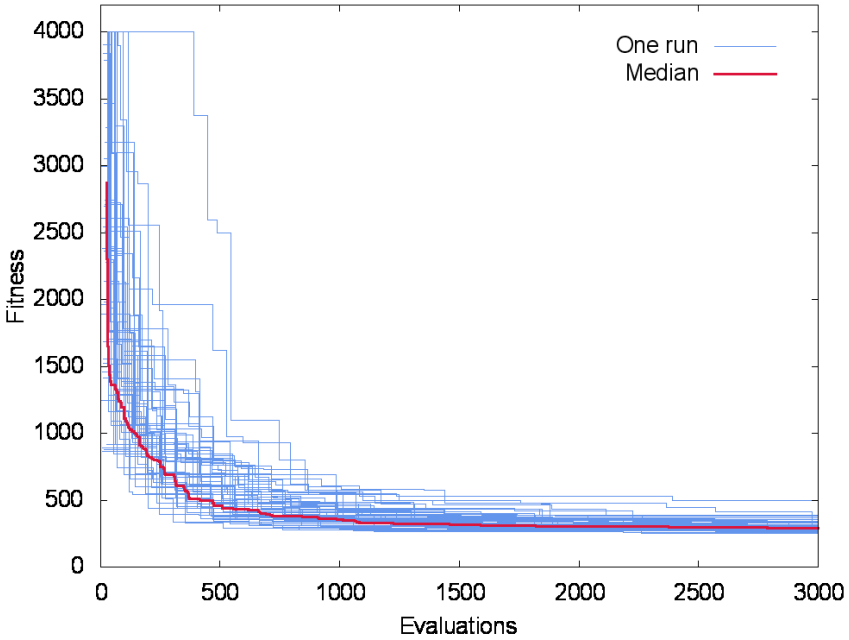


Figure 7.10: The IMRO system learning on the acrobot problem.

7.2 Real-valued outputs

The control problems studied so far have all required the controller to select from a small set of control signals ($\{-1, 1\}$ for all pole balancing problems, $\{-1, 0, 1\}$ for the acrobot problem). However, many other control problems require a real-valued control signal; it is therefore an important feature for a system to be able demonstrate the generation of successful controllers on one of these problems.

The IMRO system presented initially in Chapter 6 already includes a mechanism to produce real outputs in the range $[0, 1]$, which were used to produce the bang-zero-bang control signals required for the acrobot problem. For the acrobot problem three real outputs were used (and therefore three output genes), one for each possible action, and the action chosen was the one for which the real-valued controller output was the highest.

Here the IMRO system will be applied to the real-valued versions of the pole balancing problems on which it was tested in the previous section: the single and double pole balancing, with and without velocity inputs. In the initial randomly generated population of genomes, the IMRO output genes produce real outputs in the range $[0, 1]$, and this will be linearly mapped to the $[-10, +10]$ range required by the pole balancing problems; evolution can bring an output gene's threshold parameter outside its initial range $[-1, 1]$, which leads to gene outputs outside the range $[0, 1]$; in these cases the output will be clipped to fit in $[0, 1]$. The IMRO system setup used here is exactly the same as in the previous section, with the exception that the output gene's threshold parameter is used to generate a real output, not a boolean one. The other experimental settings, including the search algorithm, are also identical.

The results of the real-valued output of the IMRO system on these problems are shown in Table 7.4; the corresponding results of the IMRO system with 'bang-bang' control from the previous section are also included for comparison. Additionally, the following two pages show the details of the evolutionary runs on first the single pole balancing problem with and without velocity inputs (Figures 7.11 and 7.12), then on the double pole balancing problem (Figures 7.13 and 7.14).

Table 7.4: Results of the IMRO system on pole balancing problems, for bang-bang and real outputs: number of controller evaluations before a successful controller is found. Key: SD = Standard Deviation, NV = No Velocities

	IMRO - bang-bang			IMRO - real output		
	Mean(SD)	Best	Worst	Mean(SD)	Best	Worst
Single Pole	156(111)	5	589	266(186)	9	1179
Double Pole	1677(1261)	245	5719	3128(1841)	404	8646
Single Pole(NV)	2283(1433)	63	5840	1746(1262)	55	6520
Double Pole(NV)	-	-	-	-	-	-

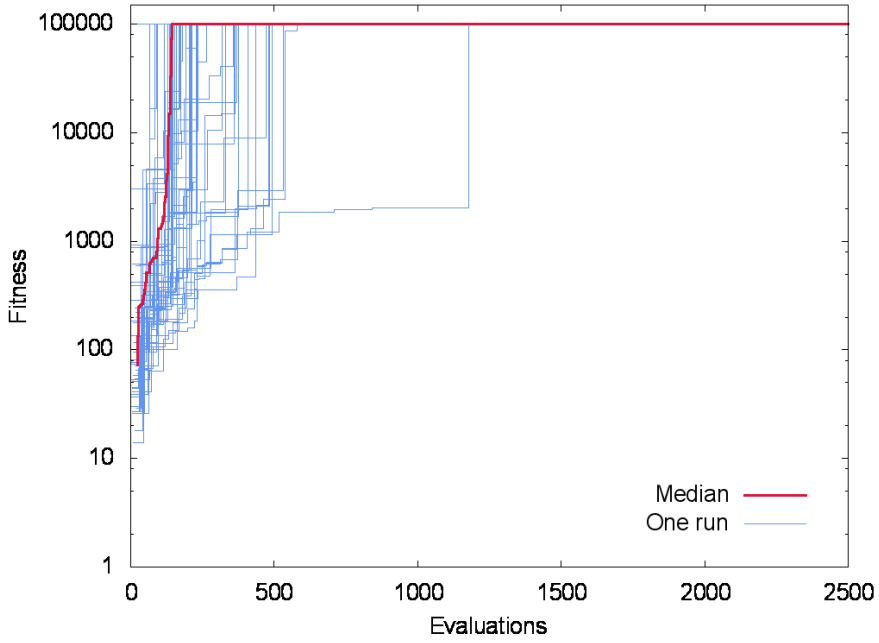


Figure 7.11: The IMRO system learning on the single pole balancing (SPB) problem with real control outputs. Each blue line represents one of the fifty runs, and the bold red line is the median of these runs at each point.

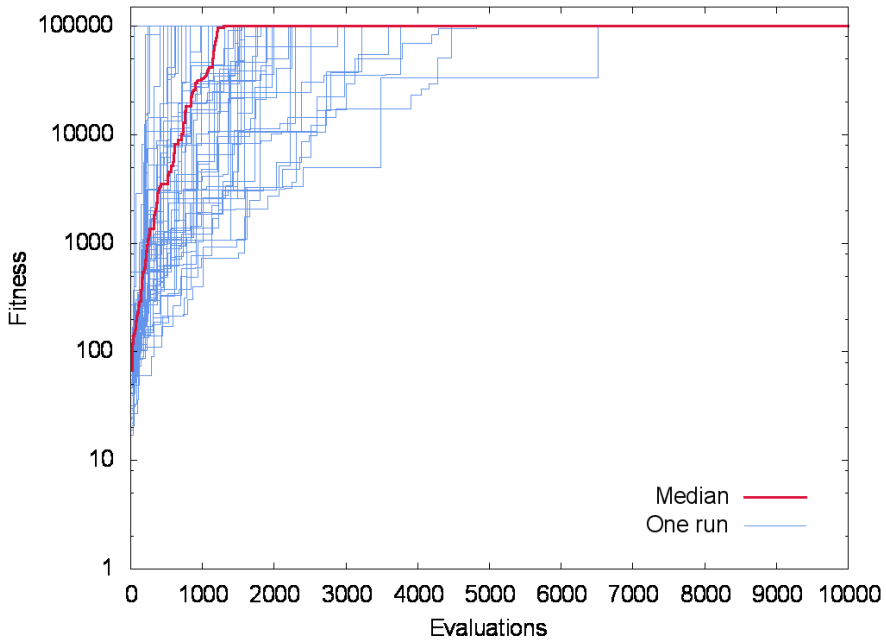


Figure 7.12: The IMRO system learning on the single pole balancing problem without velocitiy inputs (SPB(NV)) with real control outputs.

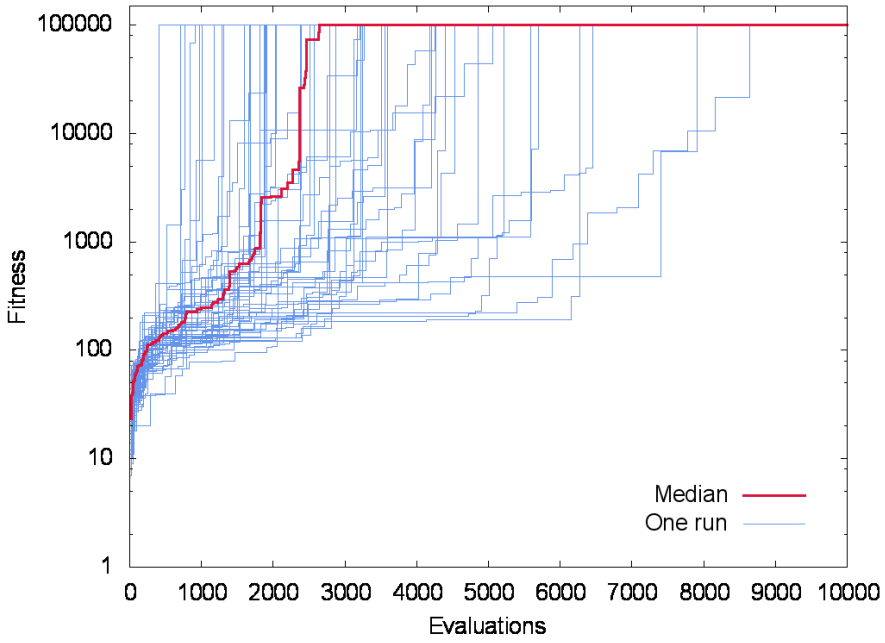


Figure 7.13: The IMRO system learning on the double pole balancing problem (DPB) with real control outputs.

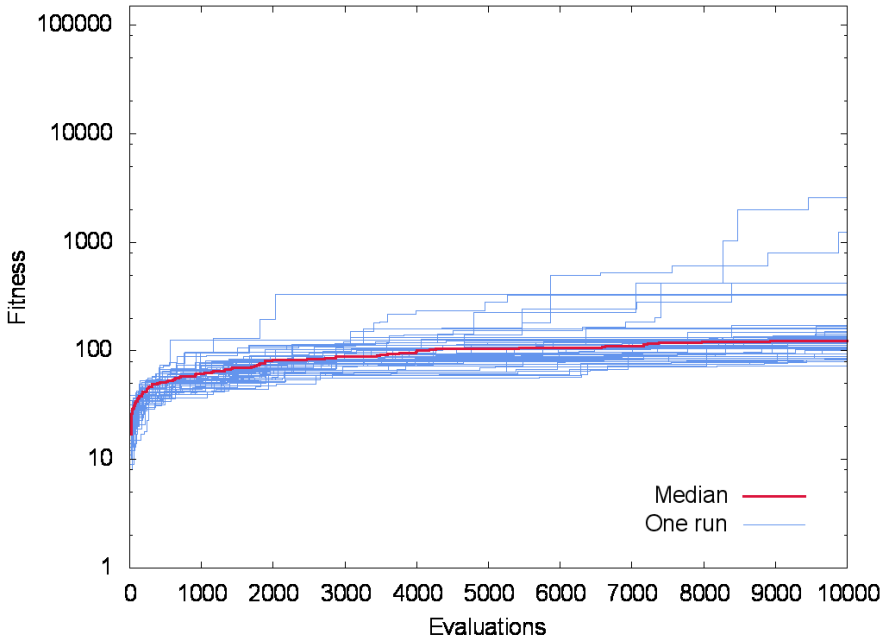


Figure 7.14: The IMRO system learning on the double pole balancing problem without velocity inputs (DPB(NV)) with real control outputs.

The number of evaluations required by the IMRO system to find a successful controller for the real output versions of the full-state single and double pole balancing problems are significantly ($p < 0.001$) higher than the number of evaluations required for the 'bang-bang' versions of the problem. However the number of evaluations required to solve the real output version of the single pole balancing problem without velocity inputs is significantly lower ($p < 0.03$) than that needed for the 'bang-bang' version of the problem. Additionally, the IMRO system was able to find successful controllers for every run of these problems within the number of evaluations imparted.

Overall, the IMRO system's performance does not diverge enough on the bang-bang and real versions of the pole balancing to justify concerns as to its suitability to real valued output control problems. The system still fails on the double pole balancing without velocity inputs, further indicating that an IMRO solution to this problem may require a complexification of some of the components of the current system.

7.3 Generalisation

The method for evaluating performance on the pole balancing problem in this thesis so far has been to assess the reliability and speed with which a successful controller is found, evaluating each candidate controller in identical conditions, which includes starting the cart and pole in the same central and balanced position. This is a common approach [GSM08], but there is another common method of evaluating performance on the pole balancing problem, introduced by Whitley et al. [WDDA93], which focuses on the tested system's ability to generalise. The differences with the methodology employed so far are:

- During learning, each controller evaluation starts with the controlled system in a different randomly generated initial state.
- When a successful controller is found, learning stops and that controller is tested on a large sample of starting positions. The number of positions from which the controller can keep the pole balanced is the generalisation score.

This method of evaluation has been used for other GRN models [NSB10][MNH$^+$12], and the same experimental settings will be used here. For each controller evaluation, a random initial state of the cart-pole system is generated, each state component being taken uniformly from the following ranges:

- The position of the cart on the track $x \in [-2.4, 2.4]m$.
- The velocity of the cart $\dot{x} \in [-1, 1]m/s$.
- The angle of the pole to the vertical $\theta \in [-12, 12]^\circ$.
- The angular velocity of the pole $\dot{\theta} \in [-1.5, 1.5]^\circ/s$.

A controller is considered successful it it balances the pole for 120,000 time steps (as opposed to 100,000 for previous experiments), but evaluations in the generalisation test are only run for at most 1,000 time steps [NSB10]. The sample of initial cart-pole states is generated by combining for each

component a subset of 5 values, giving 625 ($= 5^4$) initial states. The subset of each state component is generated by taking the values at 0.05, 0.275, 0.50, 0.725, and 0.95 of the normalised the range above associated with that component (e.g. $x \in \{-2.16, -1.08, 0, 1.08, 2.16\}$). The cart-pole system is impossible to keep balanced from some of these initial states; through exhaustive policy search, Nicolau et al. found that at least 168 out of these 625 positions cannot possibly be recovered from [NSB10], leaving a maximum possible generalisation test score of 457.

However, it should be noted that the version of the pole balancing implemented in the work of Nicolau et al. [NSB10] differs significantly from that used in this work, which is identical to that used in the work of Gomez et al. [GSM08]: the equations of motion differ (e.g. Nicolau et al.'s do not simulate friction), and the velocities in the version used by Gomez et al. are not restricted to the ranges above. This latter point especially makes them very different problems, as it makes the Gomez et al. version less stable, leading to much faster pole failure; this does not necessarily make it a harder problem, as a faster failure may lead to quicker learning. A consequence of these differences is that the generalisation scores obtained below are at times higher than what Nicolau et al. found to be the maximal score with their pole balancing dynamics; a likely explanation for this is that the bounding of the angular velocity makes irrecoverable a larger portion of the cart-pole state-space.

Following the change to random initial states (and therefore a now varying fitness for any given controller), explicit elitism (carrying over the fittest genomes from the previous generation unchanged) is disabled in the ALPS GA for these experiments; additionally the number of controller evaluations is not cut off, to provide in all cases a valid comparison between generalisation scores. All other settings are kept identical to the previous experiments in this chapter, and IMRO's generalisation abilities are tested on the single pole balancing with Bang-Bang control with this method.

Experiments are run with this new methodology on the single pole balancing with both bang-bang and real output control, for both the full state and velocity-less versions of the problem. Table 7.5 displays the results. Additionally the following two pages detail the evolutionary runs for each experiment, first the full state version (with bang-bang and real output, see Figures 7.15 and 7.16), then the velocity-less version of the problem (see Figures 7.17 and 7.18).

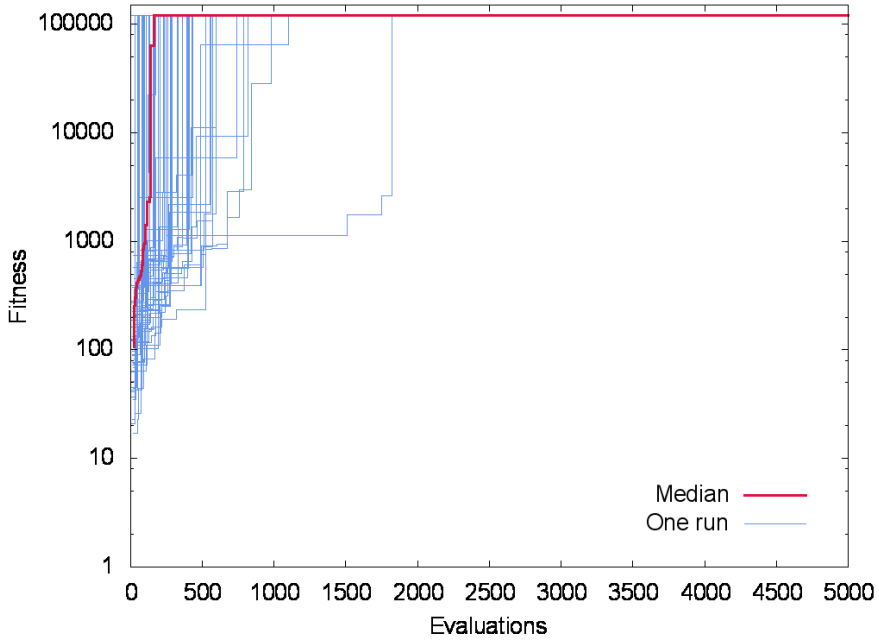


Figure 7.15: The IMRO system learning on the single pole balancing (SPB) problem with bang-bang (BB) control outputs and random state initialisation: controller evaluation starts every time in a different, random, state.

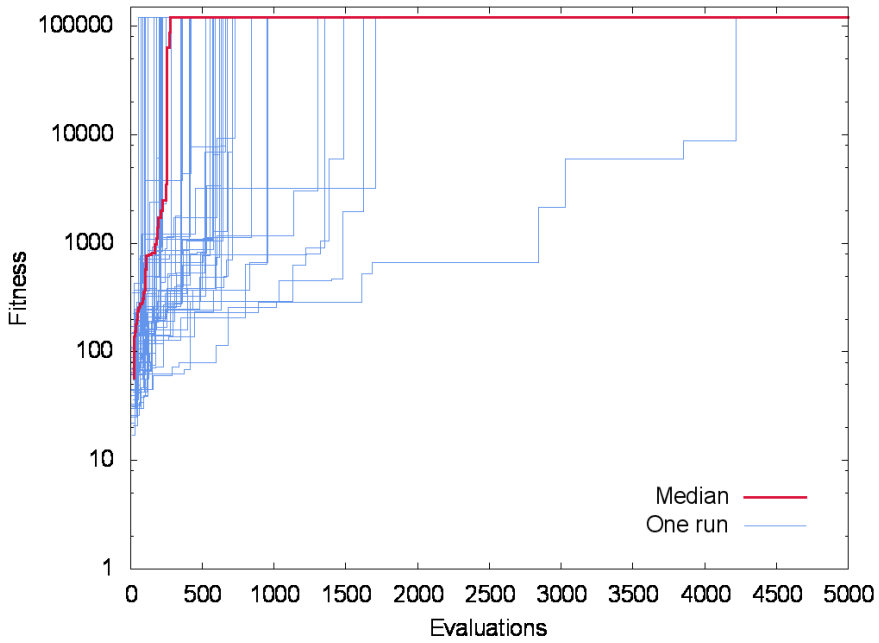


Figure 7.16: The IMRO system learning on the single pole balancing (SPB) problem with real ($\mathbb{R}$) control outputs and random state initialisation.

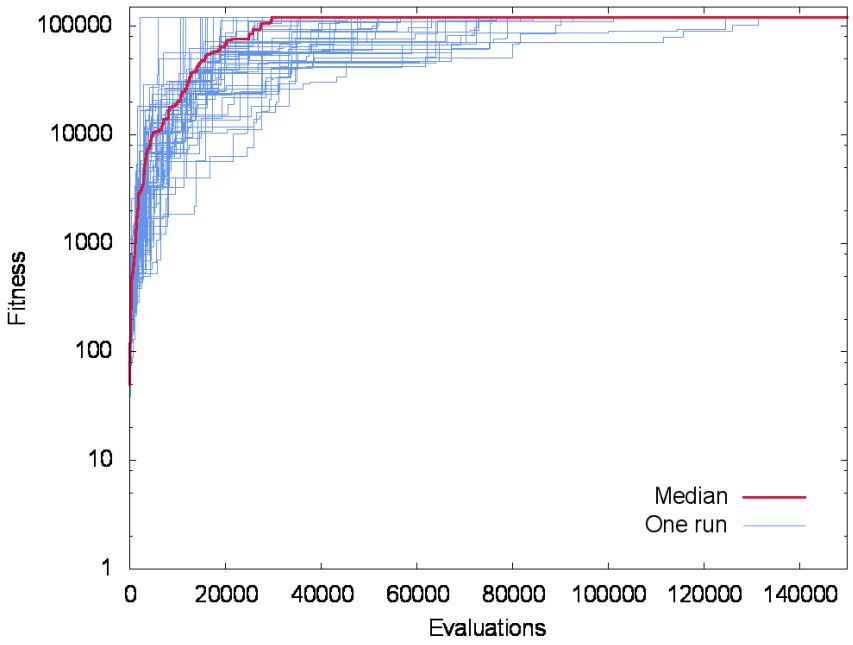


Figure 7.17: The IMRO system learning on the single pole balancing problem without velocity inputs (SPB(NV)) with bang-bang (BB) control outputs and random state initialisation. Note the large amount of evaluations required to produce a successful controller in some runs.

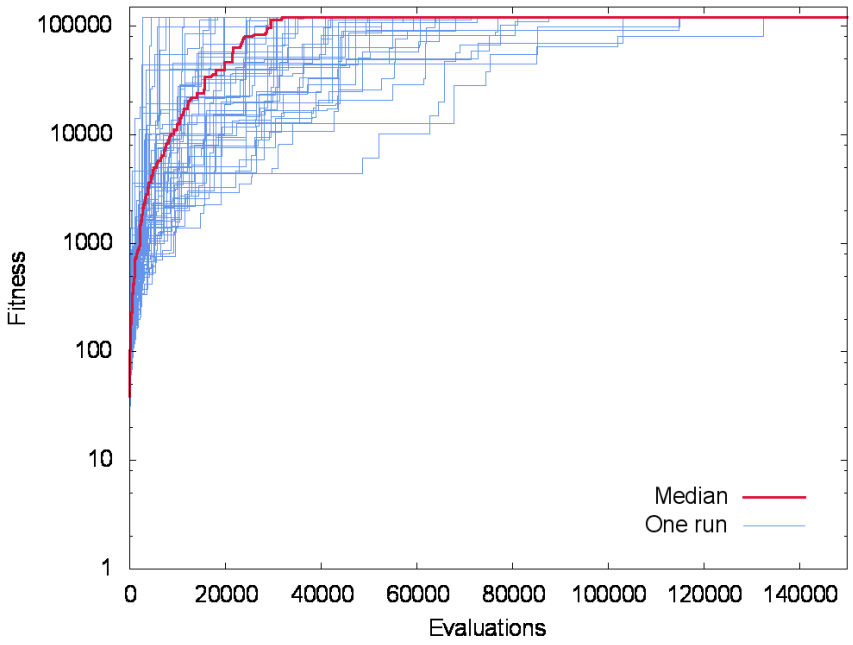


Figure 7.18: The IMRO system learning on the single pole balancing problem without velocity inputs (SPB(NV)) with real ($\mathbb{R}$) control outputs and random state initialisation.

In terms of the number of evaluations needed to find a successful controller, the performance of IMRO on the full-state version of the problem, with both bang-bang and real control output are each significantly ($p < 0.001$) worse than their respective results on the equivalent versions of the problem with a fixed, balanced and centred, initial cart-pole state. In both cases, the number of evaluations is approximately doubled which, given the change of fitness function from completely unchanging to stochastic, is not surprising.

On the other hand, using random initialisation of the cart-pole system's position and velocity seems to make the problem much harder when the velocities are withheld from the controller. A possible explanation is that when the cart-pole is always started balanced in the centre, it is possible for the controller to deduce to some degree the velocity of both the cart and pole from its own previous actions; e.g. if the controller pushes the cart to the left as a first action, it can use this as an indication that the cart's velocity is negative and the pole velocity is positive. Also controllers can then be evolved implicitly assuming that the initial velocity is null, whereas here the initial velocity is a complete unknown.

As the dynamics of the controlled systems are different, it would not be valid to infer that the IMRO system's performance is superior to Nicolau et al.'s system from these generalisation score. It is however worth noting that the mean and median generalisation scores of the IMRO system are not worse than those obtained with any of the tested versions of their system; their most successful system obtaining a mean generalisation score of 235.68, and a median score of 237 [NSB10].

Interestingly, successful controllers on problem without velocity inputs generalise as well as on average as the controllers for the full state version of the problem, but show less variation in the generalisation score. As can be seen by comparing the plots in Figures 7.16 (full-state, bang-bang) and 7.17 (no velocity inputs, bang-bang), there are very few partial solutions above 10,000 time steps for the full state version of the problem, but many for the version with no velocity inputs; an IMRO controller able to solve the full state version of the problem for 10,000 time steps is therefore very likely to also succeed on 120,000 time steps. That is not the case on the version of the problem without velocity inputs, for which many intermediate solutions exist over 10,000 time steps, on which the GA temporarily stabilises.

Table 7.5: Results for the IMRO system on the single pole balancing problem with random initial states, for bang-bang and real outputs: number of controller evaluations before a successful controller is found, and generalisation scores. Key: SD = Standard Deviation, NV = No Velocities, BB = Bang-Bang, $\mathbb{R}$ = real.

	Evaluations				Generalisation			
	Mean(SD)	Med.	Worst	Best	Mean(SD)	Med.	Worst	Best
SPB, BB	341(271)	279	4	1052	247(165)	217	0	553
SPB, $\mathbb{R}$	602(656)	526	55	4218	234(147)	277	1	545
SPB(NV), BB	47778(30257)	45452	2014	131489	237(70)	256	14	344
SPB(NV), $\mathbb{R}$	45386(30809)	38190	2792	146428	239(69)	262	86	360

Beyond lengthening the evolutionary time required to find a successful solution, this also exposes the population to more of the possible initialisation states which leads to less variation (and therefore more reliability) in the final generalisation score.

Similarly, the following evolutionary principle can be applied on the full-state version of the problem to improve the final controllers' generalisation score: Wagner showed that applying evolutionary pressure on a population past the point where an optimal solution (i.e. a solution with the maximal fitness score) as been found can increase the robustness against change of the whole population [Wag96]. In effect, the population migrates towards the 'centre' of the maximum-fitness area of the search-space which was reached by the first optimal solution. Pictorially, that maximum-fitness area constitutes a plateau bordered with cliffs; and change, particularly mutation-based change, constitutes taking a blind step in a random direction. It therefore makes sense that the genomes further away from the cliffs would have an evolutionary advantage, their offspring being less likely to fall off, even though their fitness is identical to that of a genome one step away from the fall.

This phenomenon, coupled with the added exposure to more initial states brought by a longer evolutionary run, should improve the final generalisation score. Experiments are run on the full state version of the problem, with both bang-bang and real control outputs. All experimental settings are kept identical, with the difference that the evolutionary process is ran each time for a different, additional number of evaluations after an optimal solution is found. Table 7.6 details the results.

This approach proves successful in drastically improving the mean and median generalisation score obtained by the final controller. This is most obvious for the bang-bang version of the problem, for which both an additional hundred controller evaluations, followed by an additional four hundred on top, both provide significant improvements ($p < 0.01$, and $p < 0.001$ respectively). Significant improvements can also be seen on the real output version of the problem in which the additional hundred controller evaluations significantly increase the generalisation score ($p < 0.005$).

Table 7.6: Generalisation scores of the IMRO system on the single pole balancing (SPB) with application of additional evolutionary pressure after a controller with optimal fitness is first produced. The leftmost column shows the number of additional evaluations the GA is run for after a successful controller is found. Key: SD = Standard Deviation, NV = No Velocities, BB = Bang-Bang, $\mathbb{R}$ = real.

	SPB, BB				SPB, $\mathbb{R}$			
	Mean(SD)	Median	Worst	Best	Mean(SD)	Median	Worst	Best
0	247(165)	217	0	553	234(147)	277	1	545
100	326(135)	346	14	549	313(133)	316	66	538
500	414(126)	459	0	553	330(162)	399	23	550
1000	414(123)	439	0	553	358(151)	386	14	554

However, there is little improvement in the variance; note that the controllers selected to produce the generalisation scores in Table 7.6 were for each evolutionary run the last controller evaluated with an optimal fitness. More sophisticated ways of selecting the final genome/controller may improve on this further; e.g. by making the population of final solutions with optimal fitness collapses/converges to a single point.

Or, put in the context of the previous cliff/plateau image: in each run, after applying the additional evolutionary pressure moved population of optimal solutions as a whole away from the cliff, an effectively random member of that population was selected to produce the generalisation score. This random selection means that the selected individual was sometimes still close to the cliff, as can be seen in the lack of improvement in the 'Worst' column of Table 7.6. It might therefore be preferable to take a solution towards the centre of the population, but this is hard to determine and depends on the (unknown) shape of the plateau; a more robust method may be to still take the last generated optimal individual, but to make the whole population huddle together first.

This would result in a three stage method: first, evolving a successful solution; second, keep applying evolutionary pressure for a set number of evaluations; third, make the population of optimal solutions converge (e.g. by increasing elitism, and/or reducing the mutation rate).

7.4 The mountain car problem

The IMRO system has now been successfully applied to a variety of both developmental (activation pattern generation) and control (multiple versions of the single and double pole balancing, acrobot) problems. To further test the IMRO system's applicability for control, it will here be applied to the classical mountain car control problem.

The mountain car problem is a standard control task consisting in leading an underpowered car to the top of a mountain. The car's engine is not powerful to bring it to the mountain top from a cold start, so it must take advantage of the opposite slope to build additional momentum (see Figure 7.19). Further details of the mountain car problem, including the equations of motion, are given in the literature review, Section 2.1.4.

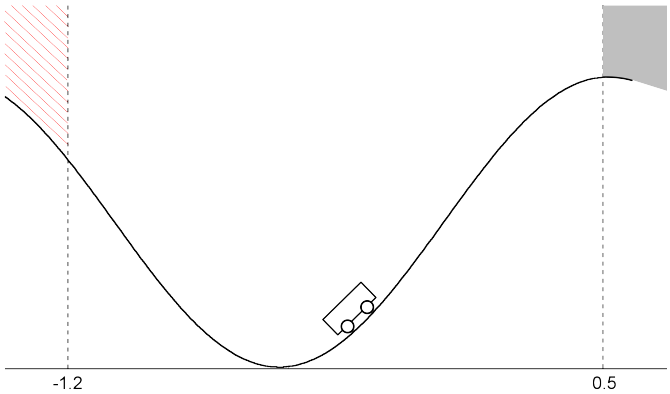


Figure 7.19: The mountain car problem. The car must reach the greyed area to the right. (Reproduction of Figure 2.6 for convenience.)

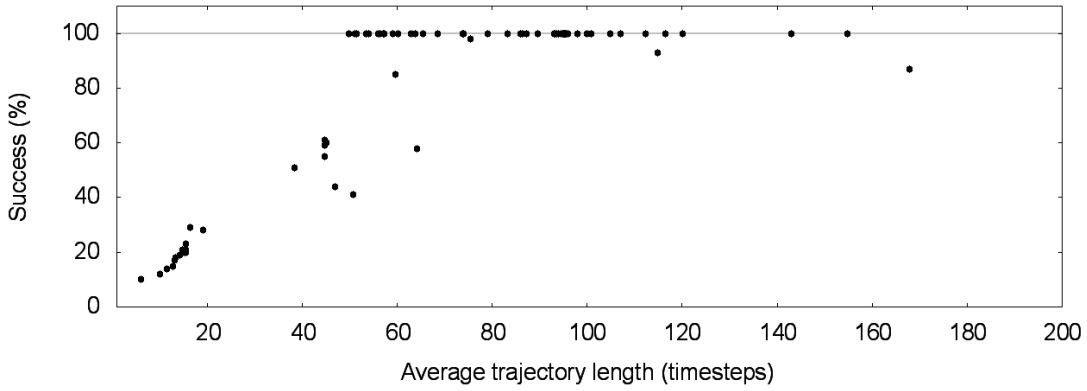


Figure 7.20: Plot of the generalisation results of the mountain car IMRO experiments. Each dot represents one run; the runs for which the dot is on the horizontal grey line on top have a perfect generalisation score: the run's final controller successfully lead the car from each of the hundred generalisation starting position/velocity pairs.

Like the acrobot problem, the mountain car problem requires 'bang-zero-bang' control, selecting at each time step one action from the set $\{-1, 0, 1\}$. The same IMRO controller setup will therefore be used here as for the acrobot in Chapter 6: three real valued outputs are produced by the controller at each time step, each corresponding to one of the possible actions, and the action with the highest output value is selected. As per the usual mountain car setup, the position and velocity inputs are normalised to the range $[0, 1]$.

Experiments are run with the GA settings identical to those of the previous section on pole balancing generalisation. The IMRO system is tested on the traditional full-state version of the mountain car problem, therefore five thousand controller evaluations are always run through.

Finding a successful controller in five thousand evaluations on this problem is not hard, as the starting position and velocity for each controller evaluation is randomly selected and sometimes such that the car is in a state very close to success. The generalisation performance is most important here.

Table 7.7: Generalisation results of the IMRO system on the classical mountain car problem. 'Success' is for each run the number generalisation position/velocity pairs on which the final controller was successful at the end of the run. The trajectory length rows give the average length of successful generalisation trajectories. Key: SD = Standard Deviation, NV = No Velocities, BB = Bang-Bang, $\mathbb{R}$ = real.

	Mean(SD)	Median	Best	Worst
Success (%)	77.2(34.5)	100	100	10
Trajectory length, all runs	67.9(37.0)	75.4	6.1	167.9
Trajectory length, successful runs	87.4(20.8)	94.6	49.9	155.0

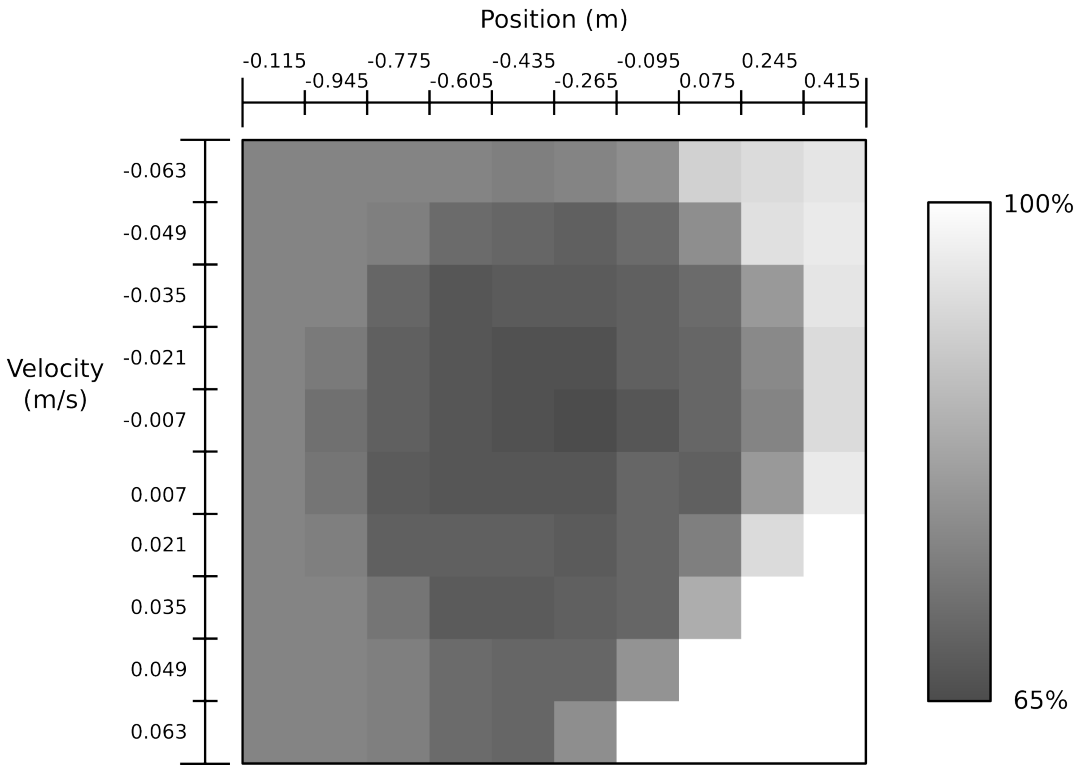


Figure 7.21: Map of the percentage of runs producing a successful final controller for each generalisation position/velocity pair. 65% of the runs produced controllers that were successful on every pair (and therefore have a perfect generalisation score). Overall it can be seen that starting states with a higher altitude (situated near the left and right borders of the map), and states with a higher absolute velocity (situated near the top and bottom borders of the map) are more likely to be solved. Particularly the states closest to the target position and with a high velocity towards that position are solved by all final controllers (as illustrated by the full white portion at the bottom right corner of the map).

To obtain a generalisation score, da Motta Salles Barreto and Anderson used the same hundred randomly generated states for multiple systems, but do not specify these states[dMSBA08]. Here, the generalisation score will be calculated with a method inspired by that of Whitley et al. [WDDA93] described in the previous section: once a successful controller is obtained, the evolutionary run will be stopped, and the controller will be tested on an array of a hundred position/velocity pairs; the generalisation score will consist of both the number of pairs for which the controller succeeded (reached the goal within a thousand time steps), and the number of time steps the controller took to reach the goal for each pair. The hundred position/velocity pairs are formed by combining two sets of ten values (one set of positions and one set of velocities). These sets are generated by taking the ten equidistant values in the normalised range $[0.05, 0.95]$; the resulting values are shown on the axes in Figures 7.21 and 7.22.

Additionally, an interesting aspect of the performance of the successful controllers will be to their behaviour from different starting positions; a hundred runs will therefore be executed instead of the fifty used before, to have enough data to allow for this analysis.

The experiment is run, and successful controllers are found in each run. Table 7.7 details the gener-

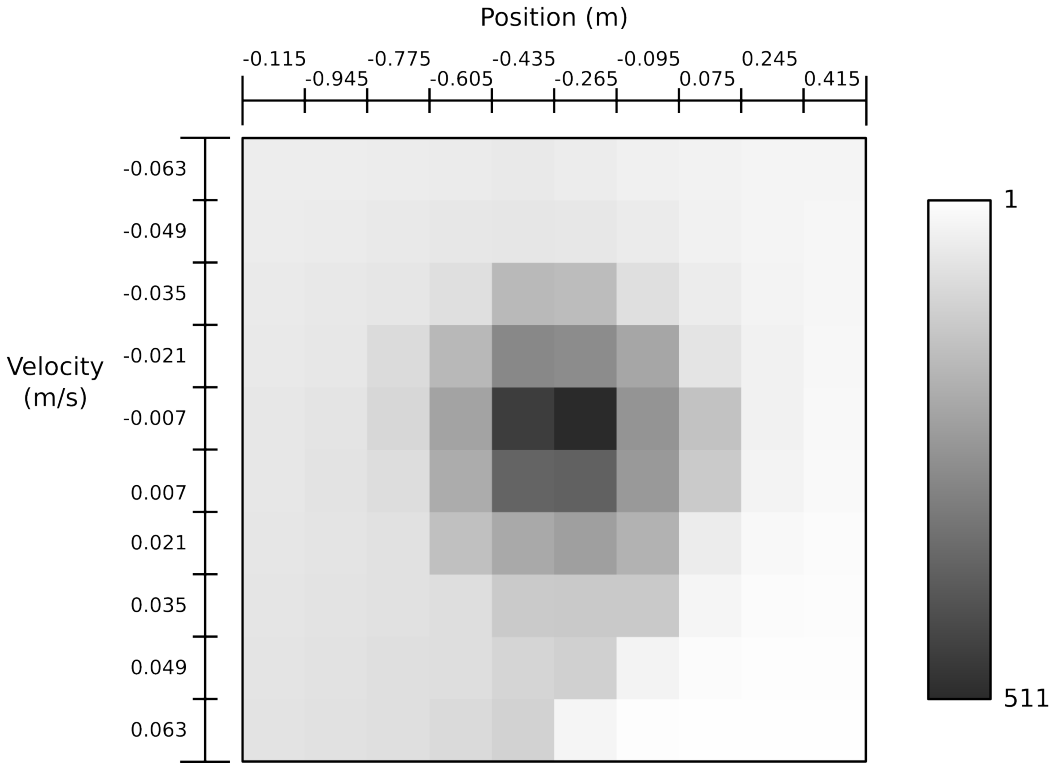


Figure 7.22: Map of the mean trajectory length for successful final controllers for each generalisation position/velocity pair. Similarly to Figure 7.21, the initial states with high absolute velocity and high altitude are be solved with shorter trajectories.

alisation performance of the final IMRO controllers, which is also further illustrated in Figure 7.20. The median success score is 100% (see Table 7.7), meaning that in most runs the final controller has a perfect generalisation score. Due to the difference in generalisation methodology with da Motta Salles Barreto and Anderson's work [dMSBA08], it is not possible to compare directly the IMRO results with theirs, though it is worth noting that the average generalisation trajectory lengths obtained here are much closer to that obtained with their most successful systems, than their less successful ones. Figures 7.21 and 7.22 provide a more detailed view of the final controllers' performance as a function of the generalisation initial position/velocity pairs.

7.5 Conclusion

In this chapter, the IMRO system was first extended to use the best performing generative protein encoding from Chapter 5. The resulting system was evaluated on developmental and control problems; overall it was not found to improve upon the performance of the original IMRO system. Therefore the rest of the chapter focused on testing various aspects of the original IMRO system.

The system produced successful controllers for the single pole balancing problem without velocity inputs, demonstrating its ability of IMRO controllers to operate with only a partial view of the controlled system's state. The system's ability to generate controllers with real valued output was then tested on the pole balancing problem with real control output; successful controllers were produced for every run

on every version of the problem previously solved by the IMRO system with bang-bang control output, and the differences in the number of evaluations required to generate a successful controller were small enough to consider this application a success.

The system's ability to generalise was tested using an alternative learning and testing methodology for the single pole balancing problem; despite the controlled system dynamics being the same, this was effectively a different problem, but the IMRO system was still able to generate successful controllers for every run, and an evolutionary principle was exploited to subsequently improve the generalisation of the final controllers. Finally, the system was applied to the classical mountain car control problem; successful controllers were found in every run and most of the final controllers had a perfect generalisation score, finding their way to the target location from every one of the tested generalisation starting states.

Overall, the IMRO system successfully passed every challenge to its applicability. The exception to this being the double pole balancing without velocity inputs; it is likely that more complex protein/protein interactions in the production of the cell state will be required to obtain successful IMRO controllers for this problem.

Also note that, with the exception of the number of input and output genes which vary with the problem requirements, all other IMRO settings have been kept identical throughout all experiments since its introduction in Chapter 6. The settings of the genetic algorithm have also been kept identical, with the single exception of the removal of elitism for problems with a stochastic controller evaluation function (i.e. the same controller could obtain very different fitnesses in different evaluations). This is an additional indication of the applicability of this method to other control problems.

Chapter 8

Conclusion

This work started from Bentley's fractal gene regulatory network (FGRN) model, a complex GRN model initially conceived for developmental purposes, the main feature of which was the generation of bitmap proteins from coordinates in the Mandelbrot fractal. This model was tested by further applying it to two developmental tasks: the generation of an arbitrary activation pattern (the binary representation of π) and of an algorithm for the calculation of π. The results were impressive, and the FGRN model was then applied to standard pole balancing problem and variations. Multiple improvements were made to the control system combining the FGRN and its associated GA, but the resulting system could not solve the double pole balancing problem.

In an attempt to further improve the FGRN model, the fractal protein encoding mechanism was studied, and limitations were identified in the expressivity of fractal proteins, leading to the introduction of simpler alternative protein encodings addressing these limitations. Statistical analyses of large samples of proteins randomly generated with each protein encoding provided insights into the effects and desirability of certain protein properties. The alternative encodings were found to perform as well or better than Bentley's fractal encoding on both developmental and control problems, but not sufficiently better to be able to tackle more difficult control problems such as the double pole balancing.

The aim was therefore switched from modifying the protein encoding to extracting the essence of the FGRN model, and removing any extraneous complexity. In this context it should be noted that the other defining characteristic of the FGRN model is the merging of the regulatory and environmental proteins into a single 'cytoplasm' which determines further activation of the behavioural and regulatory genes. Focusing on this feature, and with inspiration from the general organisation and gene roles in the FGRN, the Input-Merge-Regulate-Output (IMRO) architecture for GRN models was introduced. The IMRO architecture specifies a modular structure with well-defined inter-module interfaces. This is an important feature, allowing the independent study and improvement of separate parts of the model. For instance work to improve protein representation and merging, gene activation, and promoter matching of the merged protein product can be done simultaneously and independently and be integrated in a final model.

Simple components implementing each of these functions were introduced and computational experiments showed the resulting model displayed both quantitative and qualitative improvements. Quantitatively, the resulting IMRO model performed better than the FGRN model on a large subset of the problems to which the FGRN model was applied, in terms of both speed of convergence and the reliability with which successful solutions were found. The IMRO model was also qualitatively better in the sense that successful IMRO controllers were generated for problems for which the generation of FGRN controllers failed (the double pole balancing problem). Additionally, the first evolutionary solution of a problem previously intractable by evolutionary methods [dMSBA08] was presented (the hardened acrobot swing-up problem with 20Hz control frequency).

Another focus of this thesis has been real-world ease of use and applicability. Usability of the model was increased by both removing limitations on the input range, and removing arbitrary constants. Limitations in the input range can be troublesome in real-world use, where input data outside of the initially expected range can occur, and must then be clipped, essentially ignoring part of the data. The limitations on input range were first loosened, by allowing negative input concentrations in the FGRN model, which beyond scaling provided greater influence of inputs within the system; however the FGRN model inputs stayed limited to a fixed range. This limitation was entirely removed in the final IMRO model which can accept the full range of real-valued inputs. This is likely a much more robust approach, able to extrapolate the controller's behaviour to unexpected data. The arbitrary constants in the FGRN model (in activation, and protein production/decay) may need to be tuned to fit the particularities of a problem. In the IMRO model, those constants were removed; instead the activation and protein production functions of each gene, as well as protein decay, are now parameterised by evolvable parameters, allowing evolution to tune them as it searches for a controller.

Applicability of the models to future problems was maximised by minimising the amount of information given to the controller about the problems studied. The controller was kept ignorant of the dynamics of the problems, and the fitness functions were kept to their simplest form, being in every case the sum of reinforcements occurring in a controller evaluation. In fact, a physical implementation of the learning control system combining the GRN model and GA could literally be a 'black box' taking as only input the state of the controlled system and the reinforcement at each timestep; and a boolean reset signal on failure/success of the control system (e.g. pole falling down, acrobot having swung-up).

The IMRO model was successfully applied to a large variety of control problems covering a wide range of goals and operating conditions. Some of the problems studied required a stabilising behaviour (e.g. pole balancing), and some required oppositely to bring the controlled system to an unstable state. The model performed well when given only part of the controlled system's state, was able to produce both real and discrete control signals, and successful controllers were evolved from control runs with both fixed and randomised starting conditions.

8.1 Findings of this work

The following matters were addressed (in chronological order):

- Bentley's FGRN results on pattern activation and square root function approximation were repeated confirming the utility of the FGRN on these simple problems.
- A further demonstration of the FGRN model's evolvability was given in the approximation of π as a binary activation pattern and as an algorithm.
- For the developmental problem of generating an approximation of π, it was found that allowing the outputs of the FGRN system to influence the end phenotype in multiple ways allowed the system to reach a higher precision.
- The ability of the ALPS paradigm to reduce premature convergence was confirmed with a low significance in the case of the evolution of FGRN genomes across variations of the pole balancing problem.
- Negative input concentrations in the FGRN model were introduced and demonstrated to be usable and to improve performances on some control problems.
- The use of the behavioural activation concentration check mechanism was found to improve greatly the performance of the FGRN model on multiple variations of the pole balancing problem.
- Limitations of the fractal protein encoding mechanism were identified.
- Alternative protein encodings were introduced and found to perform as well or better than fractal protein encoding on all developmental and control problems tested.
- Statistical analyses of large samples of randomly generated proteins were run, explaining some of the causes for these differences in performance.
- Input-Merge-Regulate-Output (IMRO), a modular architecture for GRN models, was extracted from the FGRN model, clearly defining the role of each component in the controller and subdividing genes into reusable modules with well-defined interfaces.
- A simple model implementing the IMRO architecture was presented and tested on both developmental and control problems, displaying quantitative and qualitative improvements over the FGRN model. The IMRO model succeeded on the double pole balancing problem that the FGRN model was unable to solve and showed improved performances on all other problems studied.
- The first evolutionary solution to a more difficult version of the acrobot swing up problem was presented.
- A model combining the IMRO architecture with landscape protein encoding was introduced, but was not found to perform better than the simpler existing IMRO model.

- The simple IMRO model was applied to multiple variations of the pole balancing problem covering a wide range of operating conditions: real vs. discrete control signals, full vs. partial state of the controlled system as input, and fixed vs. randomised starting position.
- A known evolutionary principle was successfully applied to improve the generalisation of IMRO controllers after maximal fitness is reached.
- The IMRO model was successfully applied to the classical mountain car control problem.

8.2 Future work

The work in this thesis could be expanded in the following ways:

- The simple promoter module implemented for IMRO in Chapter 6 is a combination of a weighted sum and input-masking. As described by Schilstra and Bolouri [SB02], real cis-regulatory interactions make use of more varied operations than this allows (e.g. sigma-pi operations). The use of a sigma-pi matrix (allowing the weighted multiplication and summing of inputs) instead of a weight vector in the promoter would allow for more varied input-input interactions to determine regulatory and output gene activation. Note that, due to the modular structure of the IMRO architecture, this change could be effected by a change in only the promoter component. This change, in combination with IMRO's regime-switching, would allow the solving of harder control problems requiring these interactions. An example of such a problem could be the combined acrobot swing-up plus handstand problem for which no solution exists which does not make extensive usage of known system dynamics.
- Though this has not so far been necessary, it is possible that adding to the IMRO model the equivalent of FGRN's receptor gene (a filter which effectively reserves certain areas of the merged protein product to be used only by regulatory proteins) might also prove useful for problems requiring a large number of inputs.
- In this work all real mutation values to be added to a mutated parameter were taken from a uniform range. Randomly selecting the *scale* of the mutation first, then the mutation value, as done by Hornby [Hor09] in conjunction with the use of the ALPS GA, would likely lead to both the system being able to operate within a larger range of possible parameter values by allowing bigger mutations, and to a better ability to fine tune good solutions by making small mutations more likely. This would improve the system's ability to deal with new problems.
- The IMRO model performed better than the FGRN model on the simple pattern generation developmental problem on which it was tested. Given the FGRN was initially created for developmental purposes, it might be interesting to attempt the application of the IMRO model to more complex developmental problems. Possible useful modifications of the IMRO system for this purpose might be the addition of genes producing a constant flux of input proteins with a fixed associated value, or an input gene constantly adding short-lived proteins with ever-increasing associated values to the merged protein product to make the system's behaviour partly a function of time.

- Nicolau et al. have recently successfully applied a version modified for control of Banzhaf's artificial regulatory network (ARN), a GRN model, to algorithmic day-to-day index trading [NOB12], aiming to maximise profit. It would be interesting to know how the IMRO model performs comparatively on this problem. GRN models keep an internal state (the regulatory proteins produced during previous control timesteps still present in the system) which can greatly affect their output, and may therefore be particularly well adapted to this task, for which the optimal action to be taken at any given timestep is highly dependent on the previous state of the market.

- If for a given problem the amount of processing required to control the system is greater than that allowed by a single iteration of the IMRO system, it would be interesting to make the system run multiple iterations per control timestep. An obvious way to do this would be to run the system for a fixed number of iterations, and take as system output either the last outputs, or a combination (e.g. mean) of the outputs throughout the iterations, depending on the problem. A possible extension would be to additionally allow the system to stop before this maximum number of iterations is reached, by monitoring the fulfilment of a given condition (e.g. that an output value is above a specific threshold). This would appear promising in that it would make it possible for both evolution, and the running GRN, to have some control over the number of iterations run.

- In some control problems, a controller evaluation can be expensive. Boone's approach, described in Section 2.1.3 under 'Lookahead search', trades in additional computational costs for faster learning in terms of the number of controller evaluations required [Boo97a]. It consists in the addition of a model of the environment learning concurrently with the controllers; the model is refined at every control time step, and is used as a simulated environment to generate a new controller which determines the next action. Applied naively to the IMRO system, this means running a full genetic algorithm search at every time step, with the model acting as environment in the fitness evaluation of the candidate controllers. This would be incur high computational costs, but significant optimisations are possible to this naive approach: e.g. using the controllers found in the previous time steps to seed the population in the following time step, or only generating a new controller when the difference between the observed and simulated behaviour of the environment (as observed through the controller inputs) is higher than a fixed threshold.

  In particular, this approach may be interesting for emergency cases where fast learning of new conditions is essential. For instance when the controlled system has somehow been damaged to the point where the previously known dynamics are not valid any more, but it is important to quickly learn the new system dynamics and avoid failure.

- Chapter 7 applied an evolutionary method to IMRO which allowed to increase the generalisation score of the final controller by evolving controllers past the point where the maximum fitness was reached, This could be improved further by progressively converging the whole population to one of the maximal fitness controllers during this additional evolution period. This should ensure the selection of a final controller in a region of the fitness landscape with high *overall* fitness, and make it even more likely that the selected final controller generalises well.

The path of transforming our understanding of biological systems into models applicable to engineering problems is hard. The identification of the fundamental principles governing the workings of biological systems is strewn with difficulties, which range from over-simplification — discarding the very things we want to model — to misidentifying as essential what are only artefacts of the limitations of the biological implementation of these principles on a physical substrate. Faced with these difficulties, it has been argued we should let go of keeping full understanding of our models, and add blindly from biology what we hope to be useful features, to the point where faced with biological mechanisms we cannot reproduce, we sometimes introduce elements of complexity we do not fully grasp.

It has been the guiding principle of this thesis that we should not let go, but instead focus on implementing essential principles in a manner that still allows us to retain the ability to fully understand and to extend our models. We should be masters of our models, as this is the only way we can reasonably hope to keep on improving them and successfully integrate new features from the vast array of marvellous mechanisms Nature's ingenuity as blessed us with.

Appendix A

Algorithms details

A.1 Fgrn GA

Algorithm A.1: FGA: generating the initial random population.

```
procedure array newRandomPopulation(PopulationSize)
  declare array population := array(PopulationSize)

  for i := 1 in PopulationSize
    population[i] := randomGenome()
  end

  return population
end
```

Algorithm A.2: FGA: picking one of the two parent genomes needed to generate a child genome.

```
procedure genome pickParentGenome(population)
  declare integer index

  if rand() < RandomParentCoefficient
    index := randomInteger(PopulationSize)
  else
    index := randomInteger(ParentCoefficient * PopulationSize)
  end

  return population[index]
end
```

Algorithm A.3: FGA: ageing all genomes in the population and removing the expired ones.

```
procedure void ageAndRemoveExpired(population)
  declare array aged_population
  declare genome a_genome

  for i := 1 in PopulationSize
    a_genome := population[i]
    inc a_genome.age
    if a_genome.age < MaximumAge
      append(aged_population, a_genome)
    end
  end

  population := aged_population
end
```

Bibliography

[BA99] A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[Ban03a] W. Banzhaf. Artificial regulatory networks and genetic programming. *Genetic Programming Theory and Practice*, pages 43–62, 2003.

[Ban03b] W. Banzhaf. On the Dynamics of an Artificial Regulatory Network. In *Advances in Artificial Life: 7th European Conference (ECAL-2003)*, volume 2801 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2003.

[Ben96] P.J. Bentley. *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*. PhD thesis, University of Huddersfield, 1996.

[Ben03a] P.J. Bentley. Evolving Fractal Gene Regulatory Networks for Robot Control. In *Advances in Artificial Life: European Conference on Artificial Life (ECAL 2003)*, volume 2801 of *Lecture Notes in Computer Science (LNCS)*, pages 753–762. Springer, 2003.

[Ben03b] P.J. Bentley. Evolving Fractal Proteins. In *International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, volume 2606 of *Lecture Notes in Computer Science (LNCS)*, pages 81–92. Springer, 2003.

[Ben04a] P.J. Bentley. Evolving beyond perfection: an investigation of the effects of long-term evolution on fractal gene regulatory networks. *Biosystems*, 76(1-3):291–301, 2004.

[Ben04b] P.J. Bentley. Fractal Proteins. *Genetic Programming and Evolvable Machines*, 5:71–101, 2004.

[Ben05] P.J. Bentley. Evolving Fractal Gene Regulatory Networks for Graceful Degradation of Software. In *Self-* Properties in Complex Information Systems*, volume 3460 of *Lecture Notes in Computer Science (LNCS)*, pages 21–35. Springer, 2005.

[Ben09] P.J. Bentley. Methods for improving simulations of biological systems: systemic computation and fractal proteins. *Journal of The Royal Society Interface*, 6(Suppl 4):S451–S466, 2009.

[BGH03] N.E. Buchler, U. Gerland, and T. Hwa. On schemes of combinatorial transcription logic. *Proceedings of the National Academy of Sciences of the USA*, 100(9):51365141, 2003.

[BJ03] E. Ben-Jacob. Bacterial self-organization: co-enhancement of complexification and adaptability in a dynamic environment. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 361(1807):1283–1312, 2003.

[BLA+04] M.M. Babu, N.M. Luscombe, L. Aravind, M. Gerstein, and S.A. Teichmann. Structure and evolution of transcriptional regulatory networks. *Current Opinion in Structural Biology*, 14(3):283–291, 2004.

[BO04] A.L. Barabási and Z.N. Oltvai. Network biology: understanding the cell's functional organization. *Nature Reviews Genetics*, 5(2):101–113, 2004.

[Bon02] J.C. Bongard. Evolving modular genetic regulatory networks. In *Congress on Evolutionary Computation (CEC 2002)*, volume 2, pages 1872–1877. IEEE, 2002.

[Boo97a] G. Boone. Efficient reinforcement learning: Model-based acrobot control. In *International Conference on Robotics and Automation (ICRA 1997)*, volume 1, pages 229–234. IEEE, 1997.

[Boo97b] G. Boone. Minimum-time control of the acrobot. In *International Conference on Robotics and Automation (ICRA 1997)*, volume 4, pages 3281–3287. IEEE, 1997.

[BP97] S.C. Brown and K.M. Passino. Intelligent control for an acrobot. *Journal of Intelligent and Robotic Systems*, 18(3):209–248, 1997.

[BSA83] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[CSSM89] A. Cleeremans, D. Servan-Schreiber, and J.L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1(3):372–381, 1989.

[CW03] G.C. Conant and A. Wagner. Convergent evolution of gene circuits. *Nature genetics*, 34(3):264–266, 2003.

[Dav06] E.H. Davidson. *The regulatory genome: gene regulatory networks in development and evolution*. Academic Press, 2006.

[DKUY08] S.C. Duong, H. Kinjo, E. Uezato, and T. Yamamoto. A switch controller design for the acrobot using neural network and genetic algorithm. In *International Conference on Control, Automation, Robotics and Vision (ICARCV 2008)*, pages 1540–1544. IEEE, 2008.

[DMF06] P. Dürr, C. Mattiussi, and D. Floreano. Neuroevolution with Analog Genetic Encoding. In *Parallel Problem Solving from Nature (PPSN IX)*, volume 4193 of *Lecture Notes in Computer Science (LNCS)*, pages 671–680. Springer, 2006.

[dMSBA08] A. da Motta Salles Barreto and C.W. Anderson. Restricted gradient-descent algorithm for value-function approximation in reinforcement learning. *Artificial Intelligence*, 172(4-5):454–482, 2008.

[DRO+02] E.H. Davidson, J.P. Rast, P. Oliveri, A. Ransick, C. Calestani, C.H. Yuh, T. Minokawa, G. Amore, V. Hinman, C. Arenas-Mena, et al. A genomic regulatory network for development. *Science*, 295(5560):1669–1678, 2002.

[ED99] P. Eggenberger and R. Dravid. An evolutionary approach to pattern formation mechanisms on lepidopteran wings. In *Congress on Evolutionary Computation (CEC 1999)*, volume 1, pages 470–473. IEEE, 1999.

[ED09] D.H. Erwin and E.H. Davidson. The evolution of hierarchical gene regulatory networks. *Nature Reviews Genetics*, 10(2):141–148, 2009.

[Egg97] P. Eggenberger. Creation of neural networks based on developmental and evolutionary principles. In *International Conference on Artificial Neural Networks (ICANN 1997)*, volume 1327 of *Lecture Notes in Computer Science (LNCS)*, pages 337–342. Springer, 1997.

[Egg01] P. Eggenberger. Axonal growth in evolutionary neurogenesis. *Artificial Life and Robotics*, 5(3):137–141, 2001.

[EH03] P. Eggenberger Hotz. Genome-physics interaction as a new concept to reduce the number of genetic parameters in artificial evolution. In *Congress on Evolutionary Computation (CEC 2003)*, volume 1, pages 191–198. IEEE, 2003.

[EH04a] P. Eggenberger Hotz. Asymmetric cell division in artificial evolution. In *Congress on Evolutionary Computation (CEC 2004)*, volume 2, pages 2180–2186. IEEE, 2004.

[EH04b] P. Eggenberger Hotz. Comparing direct and developmental encoding schemes in artificial evolution: A case study in evolving lens shapes. In *Congress on Evolutionary Computation (CEC 2003)*, volume 1, pages 752–757. IEEE, 2004.

[EL03] E. Eisenberg and E.Y. Levanon. Preferential attachment in the protein network evolution. *Physical Review Letters*, 91(13):138701, 2003.

[FSS+11] J. Funkquist, V. Stephan, E. Schaffernicht, C. Rosner, and M. Berg. SOFCOM-Self-optimising strategy for control of the combustion process. *VGB PowerTech*, 3:48–54, 2011.

[GM03] F.J. Gomez and R. Miikkulainen. Active guidance for a finless rocket using neuroevolution. In *Genetic and Evolutionary Computation Conference (GECCO 2003)*, volume 2724 of *Lecture Notes in Computer Science (LNCS)*, pages 2084–2095. Springer, 2003.

[GSM08] F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *The Journal of Machine Learning Research*, 9:937–965, 2008.

[GW03] N.L. Geard and J. Wiles. A gene regulatory network for cell differentiation in Caenorhabditis elegans. In *First Australian Conference on Artificial Life*, pages 86–100. University of New South Wales, 2003.

[HJ05] J. Hallinan and P. Jackway. Network motifs, feedback loops and the dynamics of genetic regulatory networks. In *Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2005)*, pages 1–7. IEEE, 2005.

[Hor06] G.S. Hornby. ALPS: The Age-Layered Population Structure for Reducing the Problem of Premature Convergence. In *Proceedings of the 8th annual conference on Genetic and Evolutionary Computation (GECCO 2006)*, pages 815–822. ACM, 2006.

[Hor09] G.S. Hornby. Steady-state ALPS for real-valued problems. In *Proceedings of the 11th Annual conference on Genetic and Evolutionary Computation (GECCO 2009)*, pages 795–802. ACM, 2009.

[HW04a] J. Hallinan and J. Wiles. Asynchronous dynamics of an artificial genetic regulatory network. In *Artificial life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Artificial Life*, pages 399–403. MIT Press, 2004.

[HW04b] J. Hallinan and J. Wiles. Evolving genetic regulatory networks using an artificial genome. In *Second Conference on Asia-Pacific Bioinformatics (APBC 2004)*, volume 29, pages 291–296. Australian Computer Society, 2004.

[Ige03] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Congress on Evolutionary Computation (CEC 2003)*, volume 4, pages 2588–2595. IEEE, 2003.

[Jak95] N. Jakobi. Harnessing Morphogenesis. Technical Report 423, School of Cognitive and Computing Sciences, University of Sussex, 1995.

[JW10] M. Joachimczak and B. Wróbel. Evolving gene regulatory networks for real time control of foraging behaviours. In *Artificial Life XII : Proceedings of the Twelfth International Conference on the Simulation and Synthesis of Living Systems*, pages 348–355. MIT Press, 2010.

[Kau69] S.A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22(3):437–467, 1969.

[Kau93] S.A. Kauffman. *The Origins of Order*. Oxford University Press, 1993.

[KB03] S. Kumar and P.J. Bentley. Biologically Inspired Evolutionary Development. In *International Conference on Evolvable Systems: From Biology to Hardware (ICES 2003)*, volume 2606 of *Lecture Notes in Computer Science (LNCS)*, pages 57–68. Springer, 2003.

[KB04] P.D. Kuo and W. Banzhaf. Small World and Scale–Free Network Topologies in an Artificial Regulatory Network Model. In *Artificial life IX: proceedings of the Ninth International Conference on the Simulation and Synthesis of Artificial Life*, pages 404–409. The MIT Press, 2004.

[KFOY05] K. Kawada, S. Fujisawa, M. Obika, and T. Yamamoto. Creating swing-up patterns of an acrobot using evolutionary computation. In *Computational Intelligence in Robotics and Automation (CIRA 2005)*, pages 261–266. IEEE, 2005.

[KLB04] P.D. Kuo, A. Leier, and W. Banzhaf. Evolving dynamics in an artificial regulatory network model. In *Parallel Problem Solving from Nature (PPSN VIII)*, volume 3242 of *Lecture Notes in Computer Science (LNCS)*, pages 571–580. Springer, 2004.

[KNS06] J.F. Knabe, C.L. Nehaniv, and M.J. Schilstra. Evolutionary robustness of differentiation in genetic regulatory networks. In *Explorations in the Complexity of Possible Life; Proceedings of the 7th German Workshop on Artificial Life (GWAL7)*, pages 75–84. IOS Press, 2006.

[KNSQ06] J.F. Knabe, C.L. Nehaniv, M.J. Schilstra, and T. Quick. Evolving Biological Clocks using Genetic Regulatory Networks. In *Artificial life X: proceedings of the Tenth International Conference on the Simulation and Synthesis of Living Systems*, volume 10, pages 15–21. MIT Press, 2006.

[KSN08] J.F. Knabe, M.J. Schilstra, and C. Nehaniv. Evolution and Morphogenesis of Differentiated Multicellular Organisms: Autonomously Generated Diffusion Gradients for Positional Information. In *Artificial Life XI*, volume 11, pages 321–328. MIT Press, 2008.

[Kum05] S. Kumar. A Developmental Genetics-Inspired Approach to Robot Control. In *Proceedings of the 2005 workshops on Genetic and Evolutionary Computation (GECCO 2005)*, pages 304–309. ACM, 2005.

[LC11] R. Lopes and E. Costa. ReNCoDe: a regulatory network computational device. In *Genetic Programming*, volume 7491 of *Lecture Notes in Computer Science (LNCS)*, pages 142–153. Springer, 2011.

[Lin68a] A. Lindenmayer. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology*, 18(3):280–299, 1968.

[Lin68b] A. Lindenmayer. Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs. *Journal of theoretical biology*, 18(3):300–315, 1968.

[LKB07] A.E. Leier, P.D. Kuo, and W. Banzhaf. Analysis of preferential network motif generation in an artificial regulatory network model created by duplication and divergence. *Advances in Complex Systems*, 10(2):155–172, 2007.

[LP03] M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[Mat05] C. Mattiussi. *Evolutionary Synthesis of Analog Networks*. PhD thesis, cole Polytechnique Fdrale de Lausanne (EPFL), 2005.

[MEKK08] J.H. Metzen, M. Edgington, Y. Kassahun, and F. Kirchner. Evolving Neural Networks for Online Reinforcement Learning. In *Parallel Problem Solving from Nature (PPSN X)*, volume 5199 of *Lecture Notes in Computer Science (LNCS)*, pages 518–527. Springer, 2008.

[MF07] C. Mattiussi and D. Floreano. Analog Genetic Encoding for the Evolution of Circuits and Networks. *Transactions on Evolutionary Computation*, 11(5):596–607, 2007.

[MH01] A.F.M. Marée and P. Hogeweg. How amoeboids self-organize into a fruiting body: Multicellular coordination in Dictyostelium discoideum. *Proceedings of the National Academy of Sciences*, 98(7):3879–3883, 2001.

[MNH+12] E. Murphy, M. Nicolau, E. Hemberg, M. ONeill, and A. Brabazon. Differential Gene Expression with Tree-Adjunct Grammars. In *Parallel Problem Solving from Nature (PPSN XII)*, volume 7491 of *Lecture Notes in Computer Science (LNCS)*, pages 377–386. Springer, 2012.

[Moo91] A. Moore. Variable resolution dynamic programming: Efficiently learning action maps in multivariate real-valued state-spaces. In *Machine Learning: Proceedings of the Eighth International Conference*. Morgan Kaufmann, 1991.

[MSR91] E. Mjolsness, DH Sharp, and J. Reinitz. A Connectionist Model of Development. *Journal of theoretical Biology*, 152(4):429–453, 1991.

[NOB12] M. Nicolau, M. ONeill, and A. Brabazon. Applying Genetic Regulatory Networks to Index Trading. In *Parallel Problem Solving from Nature (PPSN XII)*, volume 7492 of *Lecture Notes in Computer Science (LNCS)*, pages 428–437. Springer, 2012.

[NSB10] M. Nicolau, M. Schoenauer, and W. Banzhaf. Evolving Genes to Balance a Pole. In *Genetic Programming; 13th European Conference (EuroGP 2010)*, volume 6021 of *Lecture Notes in Computer Science (LNCS)*, pages 196–207. Springer, 2010.

[PC07] S. Patel and C.D. Clack. ALPS evaluation in financial portfolio optimisation. In *Congress on Evolutionary Computation (CEC 2007)*, pages 813–819. IEEE, 2007.

[QNDR03] T. Quick, C.L. Nehaniv, K. Dautenhahn, and G. Roberts. Evolving Embodied Genetic Regulatory Network-Driven Control Systems. In *Advances in Artificial Life; 7th European Conference (ECAL 2003)*, volume 2801 of *Lecture Notes in Computer Science (LNCS)*, pages 266–277. Springer, 2003.

[Rei99] T. Reil. Dynamics of Gene Expression in an Artificial Genome - Implications for Biological and Artificial Ontogeny. In *5th European Conference on Advances in Artificial Life (ECAL 1999)*, volume 1674 of *Lecture Notes in Computer Science (LNCS)*, pages 457–466. Springer, 1999.

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. Cambridge University Press, 1998.

[SB02] M.J. Schilstra and H. Bolouri. Modelling the regulation of gene expression in genetic regulatory networks. Technical report, BioComputation group, University of Hertfordshire, 2002.

[SF99] H. Shimooka and Y. Fujimoto. Generating Equations with Genetic Programming for Control of a Movable Inverted Pendulum. In *Simulated Evolution and Learning (SEAL 1998)*, volume 1585 of *Lecture Notes in Computer Science (LNCS)*, pages 179–186. Springer, 1999.

[SOMMA02] S.S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetics*, 31(1):64–68, 2002.

[Spo95] M.W. Spong. The swing up control problem for the acrobot. *Control Systems*, 15(1):49–55, 1995.

[Sut96] R.S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS 1996)*, pages 1038–1044. MIT Press, 1996.

[TB04] S.A. Teichmann and M.M. Babu. Gene regulatory network growth by duplication. *Nature Genetics*, 36(5):492–496, 2004.

[Wag94] A. Wagner. Evolution of gene networks by gene duplications: a mathematical model and its implications on genome organization. *Proceedings of the National Academy of Sciences of the USA*, 91(10):4387–4391, 1994.

[Wag96] A. Wagner. Does evolutionary plasticity evolve? *Evolution*, 50(3):1008–1023, 1996.

[WDDA93] D. Whitley, S. Dominic, R. Das, and C.W. Anderson. Genetic Reinforcement Learning for Neurocontrol Problems. *Machine Learning*, 13(2):259–284, 1993.

[Wie90] A.P. Wieland. Evolving Controls for Unstable Systems. In *Connectionist Models: Proceedings of the 1990 Summer School*, pages 91–102. Morgan Kaufman, 1990.

[Wue98] A. Wuensche. Genomic regulation modeled as a network with basins of attraction. In *Pacific Symposium on Biocomputing*, volume 3, pages 89–102. World Scientific Publishing, 1998.

[YBD01] C.H. Yuh, H. Bolouri, and E.H. Davidson. Cis-regulatory logic in the endo16 gene: switching from a specification to a differentiation mode of control. *Development*, 128(5):617–629, 2001.

[YNTI05] J. Yoshimoto, M. Nishimura, Y. Tokita, and S. Ishii. Acrobot control by learning the switching of multiple controllers. *Artificial Life and Robotics*, 9(2):67–71, 2005.

[Zah11] P. Zahadat. *Evolution and Development in Gene Regulatory Systems*. PhD thesis, Shiraz University, 2011.

[ZCS+10] P. Zahadat, D.J. Christensen, U. P. Schultz, S.D. Katebi, and K. Stoy. Fractal Gene Regulatory Networks for Robust Locomotion Control of Modular Robots. In *From Animals to Animats 11: 11th International Conference on Simulation of Adaptive Behavior (SAB 2010)*, volume 6226 of *Lecture Notes in Computer Science (LNCS)*, pages 544–554, 2010.

[ZK08] P. Zahadat and S.D. Katebi. Tartarus And Fractal Gene Regulatory Networks With Inputs. *Advances in Complex Systems (ACS)*, 11(06):803–829, 2008.

[ZS12] P. Zahadat and K. Stoy. An alternative representation of fractal gene regulatory networks facilitating analysis and interpretation. *Annals of Mathematics and Artificial Intelligence*, 65(4):285–316, 2012.

[ZSC12] P. Zahadat, T. Schmickl, and K. Crailsheim. Evolving reactive controller for a modular robot: Benefits of the property of state-switching in fractal gene regulatory networks. In *From Animals to Animats 12: 12th International Conference on Simulation of Adaptive Behavior (SAB 2012)*, volume 7426 of *Lecture Notes in Computer Science (LNCS)*, pages 209–218. Springer, 2012.