Shiraz University
Faculty of Engineering

Ph.D. Dissertation in Computer Engineering (Artificial Intelligence)

EVOLUTION AND DEVELOPMENT IN GENE
REGULATORY NETWORK SYSTEMS

By
PAYAM ZAHADAT

Supervised by
Prof. Seraj D. Katebi

February 2011
ACKNOWLEDGMENTS

First, I want to thank my supervisor Dr. Serajeddin Katebi for his supports and constructive guidance during my PhD. Also, I would like to thank other members of my thesis committee Dr. Gholamhosein Dastgheibifard, Dr. Kourosh Ziarati, and Dr. Ebrahim Farjah. I also thank Dr. Kasper Stoy for giving me a great stay with his group in Modular Robotics Lab in University of Southern Denmark, and for his supports and guidance. I want to offer special thanks to all my colleagues in Shiraz University and SDU for their helps and friendship. In addition I particularly thank my family for their love, support, and encouragements in all my life. And I would like to express my gratitude to everyone who gave me their time, patience, knowledge and friendship. Thanks all of you!
ABSTRACT

EVOLUTION AND DEVELOPMENT IN GENE REGULATORY NETWORK SYSTEMS

BY

PAYAM ZAHADAT

Gene Regulatory Networks (GRN) are the main control part of the metabolic systems in the real world. While a genome in a cell is a blue-print of the whole organism, development of the structure and behaviors of the system is determined through ongoing interactions between the genes and the proteins which are encoded in the genome. Metabolic GRNs are evolved during a long process of evolution and drive the complex living behaviors of the successful organisms in the world. In the field of evolutionary computations, researchers seek for the proper methods and representations to improve the evolvability of the systems in order to solve complex problems. Computational GRNs are introduced in order to help understanding of the real mechanisms of biological GRNs and also to make complex phenotypes from relatively simple genotypes in the engineering fields. This work briefly describes different aspects and models of several computational GRNs and investigates the concepts and applications of a particular GRN model called Fractal Gene Regulatory Network (FGRN). The basic model is modified in order to evolve proper solutions for the problems and investigate the behaviors of the system in special situations. Various experiments are performed and applications of the model are investigated for different problems. Results are demonstrated and discussed in detail during the thesis.
# TABLE OF CONTENTS

1 Introduction 1

2 A Biological Overview 5
   2.1 Introduction ........................................... 6
   2.2 Cell ................................................. 6
   2.3 Genome ............................................... 7
   2.4 Protein ............................................... 8
   2.5 Development of an organism .......................... 12
      2.5.1 Communication ..................................... 16
      2.5.2 Regulation activities ............................. 17

3 Literature Survey 18
   3.1 Introduction ........................................... 19
   3.2 Various aspects of GRNs in different computational models ... 20
   3.3 Kauffman’s model ........................................ 23
   3.4 Dellaert’s model ........................................ 24
   3.5 Reil’s model ............................................ 26
   3.6 Jacobi’s model .......................................... 27
   3.7 Eggenberger’s model .................................... 30
   3.8 Bongard’s model ........................................ 32
   3.9 Banzhaf’s model ........................................ 36
   3.10 Taylor’s model ......................................... 39

4 Fractal Gene Regulatory Networks 42
   4.1 Introduction ............................................ 43
      4.1.1 Abstract concepts from biology ....................... 43
   4.2 FGRN system ............................................. 44
   4.3 Genes .................................................. 45
   4.4 Fractals in an FGRN system ............................. 47
4.5 Interactions of fractal proteins ............................. 49
  4.5.1 Merging ................................................. 49
  4.5.2 Masking ................................................. 50
  4.5.3 Matching ................................................. 50
4.6 Development of a cell ......................................... 51
4.7 Evolving an FGRN system ...................................... 53
4.8 Various aspects of GRNs in FGRN model ..................... 54

5 Case studies: Sensor-coupled Fractal Gene Regulatory Net-
works for a Path Planning Problem ............................. 55
  5.1 Introduction ............................................... 56
  5.2 Problem Description ........................................ 56
  5.3 Computational Procedure .................................... 57
    5.3.1 Previous methods ...................................... 57
    5.3.2 Proposed method ...................................... 59
  5.4 Simulation Results and Discussion ........................... 66
    5.4.1 Progress of the Algorithm .............................. 67
    5.4.2 Effects of input information ............................ 70
  5.5 Conclusion ............................................... 74

6 Case studies: Investigating Evolutionary Behavior of Fractal
Gene Regulatory Networks in Presence of the Best Solution
from the Past Generations ......................................... 76
  6.1 Introduction ............................................... 77
  6.2 Problem Description ........................................ 77
  6.3 Computational Procedure .................................... 78
  6.4 Simulation Results and Discussion ........................... 81
    6.4.1 First Experiment ...................................... 82
    6.4.2 Second Experiment ...................................... 82
    6.4.3 Third Experiment ...................................... 83
    6.4.4 Which proteins are effective on a gene in a specific de-
    velopmental cycle? ......................................... 83
    6.4.5 Imitational behavior emerges: While the fittest found
    pattern is visible, what technique the GRN individuals
    employ to solve the problem? .............................. 84
  6.5 Conclusion ............................................... 89
7 Case studies: Fractal Gene Regulatory Networks for Robust Control of Modular Reconfigurable Robots 90

7.1 Control of Modular Robots ........................................... 91
  7.1.1 Introduction ....................................................... 91
  7.1.2 Modular Self-Reconfigurable Robots ....................... 91
  7.1.3 ATRON Robot architecture .................................... 93
  7.1.4 Robot simulator .................................................. 94
  7.1.5 Control in a Modular Robot .................................... 95
  7.1.6 Implementing FGRN to control modular robots ............ 96

7.2 Fractal Gene Regulatory Networks for Robust Locomotion of Modular Robots .................................................... 98
  7.2.1 Problem Description ............................................ 98
  7.2.2 Computational Procedure ...................................... 99
  7.2.3 Simulation Results and Discussion .......................... 100
    7.2.3.1 Genetic and developmental configurations .............. 100
    7.2.3.2 Case studies ................................................ 100
    7.2.3.3 Is the FGRN system still evolvable? ...................... 102

7.3 Sensor-coupled Fractal Gene Regulatory Networks for Locomotion Control of a Modular Snake Robot ...................... 104
  7.3.1 Problem Description ............................................ 104
  7.3.2 Computational Procedure ...................................... 105
  7.3.3 Simulation Results and Discussion .......................... 106
    7.3.3.1 Evolving controllers with unrestricted access sensors .................................................. 107
    7.3.3.2 FGRN with restricted sensor information .............. 109
    7.3.3.3 Comparison of behaviors from the best evolved controllers of the two experiments .............. 111

7.4 Conclusion ......................................................... 113

8 Conclusion ............................................................. 115

A Mandelbrot Fractal Set .................................................. 118
  A.1 Introduction ........................................................ 118
  A.2 Fractals in nature .................................................. 118
  A.3 Mandelbrot fractal set ............................................ 121
  A.4 Generating the Mandelbrot fractal ............................... 121
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>a) An example prokaryotic cell; b) An example Eukaryotic cell</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>DNA consists of two complementary strands wound around each other to form a double helix</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Proteins are greatly different in size, shape, and function</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Amino acids</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>a) Protein hierarchical levels; b) Protein functions</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>The coded information in DNA is converted into the amino acid sequences of proteins by a multistep process</td>
<td>14</td>
</tr>
<tr>
<td>2.7</td>
<td>DNA replication</td>
<td>14</td>
</tr>
<tr>
<td>3.1</td>
<td>An example of GRN.</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>State space of an RBN with 10 nodes (self connections are not shown).</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>A gene.</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>A protein has been rotated until the match between the link template on the gene and the corresponding protein segment goes over a certain threshold. The contribution this protein makes to the regulation of the gene is then calculated from matching the diametrically opposite side of the protein with an arbitrary fixed template.</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Different regulation schemes: a) intracellular regulation. b) intercellular regulation. c) The intercellular communication through on a receptor which depends on the affinity between the receptor and the regulatory protein</td>
<td>30</td>
</tr>
</tbody>
</table>
3.6 A schematic representation, from the side, of three structural units of an agent are shown above. Four of the six gene product diffusion sites are shown, the other two lie at the top and bottom of the spherical units. All three units contain a touch sensor neuron, (TS) but the motor neuron (M) in the first unit was deleted by gene product diffusion. The genome of the agent is displayed, along with parameter values for two genes.  
3.7 The genotype-phenotype mapping.  
3.8 Interaction between genome and proteins  
3.9 An example genome. The start sequence for coding region is 010, the start sequence for enhancer is 12 and for inhibitor is 23. En and In are labels for enhancers and inhibitors, respectively, for Gene n. So the expression of Gene 1, for example, is enhanced by protein 20 (110 in base-4 notation), and inhibited by protein 8 (020 in base-4 notation).  
3.10 The structure of a gene. In this example, the gene produces protein 48 (300 in base 4) when expressed. It has a Gradient Above Zero output function (2) with gradient 3 (03). The product is deposited at a specific diffusion site (3), which is site number 2 (02).  
3.11 Summary of the GRN Controller design. For clarity, only four of the eight diffusion sites are shown.  
4.1 An example cell (left) and a gene (right).  
4.2 A gene consists of a promoter region, a coding region, and some parameters.  
4.3 a) An example of a square window on Mandelbrot fractal set and the three values which specify it; b) The related bitmap image of the fractal protein.  
4.4 Two bitmap square (a) and (b) are merged and make (c).  
4.5 Bitmap square (a) is masked by bitmap square (b) and makes (c).  
4.6 Bitmap square (a) matches against square (b) and make (c).  
4.7 Interaction in an FGRN cell, in every developmental cycle.  
5.1 a) An invalid board configuration, b) A valid board configuration, c) The agent perception
5.2 DNA consists of two complementary strands wound around each other to form a double helix ........................................ 57
5.3 Behavioral genes are separated into two different sets based on their subtype ID; calculated values of genes inside each set are summed and produce two outputs ($Output_a$, $Output_d$). Outputs determine robot action .................................................. 60
5.4 Sensory information issue input signals which activate corresponding input genes ................................................. 62
5.5 Structure of a gene .......................................................... 63
5.6 Complete procedure .......................................................... 65
5.7 Input genes with subtype ID of number $2i$ and number $2i+1$ are related to sensor[i] ......................................................... 67
5.8 Best fitness versus generation number ................................... 68
5.9 a) Selected board configuration, b) final board configuration, c) command sequence issued by the solution ......................... 68
5.10 Sequence of concentration levels of the matching between cytoplasm content and promoter of each behavioral gene ........ 69
5.11 Sequence of produced outputs and corresponding robot command sequence ......................................................... 69
5.12 Concentration level of regulatory proteins corresponding to each regulatory gene ......................................................... 69
5.13 Activation pattern of input genes ........................................ 70
5.14 a) The sequence of concentration level of matching between cytoplasm content and the promoters of the behavioral genes, b) the sequence of produced outputs and robot command, c) the sequence of concentration level of the regulatory proteins of the cell4 ......................................................... 71
5.15 a) The sequence of concentration level of matching between cytoplasm content and the promoters of the behavioral genes, b) the sequence of produced outputs and robot command, c) and the sequence of concentration level of the regulatory proteins of the cell0 ......................................................... 72
5.16 Input proteins of cell4 ......................................................... 72
5.17 Input proteins of cell0 ......................................................... 73
5.18 Receptor, maternal, and effective areas of the input proteins of cell4 ......................................................... 73
5.19 Receptor, maternal, and effective areas of the input proteins of cell0

6.1 Complete procedure.

6.2 Fitness curves of the simple FGRN and the FGRN sensing the fittest pattern.

6.3 Fitness curve of the FGRN sensing the fittest pattern (fitness/10), effectiveness curve of input genes on behavioral genes, and effectiveness curve of regulatory genes on behavioral genes.

6.4 Imitation error and fitness curve.

6.5 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, fittest generated pattern represented to the individual, and the difference between the generated pattern and the environmental pattern for the first selected individual.

6.6 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, desired pattern represented to the individual, and the difference between the generated pattern and the environmental pattern for the first selected individual.

6.7 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, randomly generated pattern presented to the individual, and the difference between the generated pattern and the environmental pattern for the first selected individual.

6.8 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, fittest generated pattern presented to the individual, and the difference between the generated pattern and the environmental pattern for the second selected individual.

6.9 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, desired pattern presented to the individual, and the difference between the generated pattern and the environmental pattern for the second selected individual.
6.10 Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, desired pattern represented to the individual, and the difference between the generated pattern and the environmental pattern for the first selected individual. 

7.1 Examples of different modular robots 
7.2 An ATRON module 
7.3 Different morphologies of ATRON robot 
7.4 a) Different cells develop from identical genome, b) Different watermelons with the same genetic code, are grown in different environments 
7.5 An identical version of FGRN genome is copied to all the local controllers of the modules 
7.6 Different morphologies of the ATRON robot in the locomotion task. 
7.7 Velocity of the Quadrupedal, Crawler, and the two-Wheeler robots. 
7.8 Velocity of the crawler robot before a module failure, after failure, and after more evolutionary generations. 
7.9 a) A seven-segment snake ATRON robot; b) Sensor values of the modules in the initial state of the robot 
7.10 An ATRON module with tilt sensors. 
7.11 One developmental cycle. 
7.12 Each module contains an FGRN controller that specifies the actuator’s absolute position. 
7.13 Internal dynamics of the controllers of the first experiment for the two first modules of the snake. Green lines represent sensor value, black lines represent the output for the actuator absolute position, and the gray lines represent the actuator’s real position. 
7.14 Internal dynamics of the selected controller of the second experiment for the two first modules. Green lines represent the sensor values, black lines represent the output for the actuator absolute position; and the red and brown lines represent the concentration level of the two regulatory proteins.
7.15 Actuators’ absolute positions of all the modules for a typical controller of the experiments with unrestricted (a) and restricted (b) access to the sensor values. ................................. 112

A.1 Fractals can be found everywhere in nature. a) Fern; b) A cross between broccoli and cauliflower; c) Growth pattern on a leaf; d) Chaotic branching pattern from electrical discharge; e) Rivers in Savannah Georgia; f) Natural claud spiral; g) Bacterial colony; h) Another bacterial colony .......................................................... 120

A.2 For different values of $C$, this magnitude either stays equal to or below 2 forever, or it will eventually surpass 2. Once the magnitude of $Z$ surpasses 2, it will increase forever. ....................... 122

A.3 Mandelbrot fractal set. ................................................................. 123
LIST OF TABLES

3.1 Phenotypic effect of neural development gene products The rewrite rules are triggered when the gene product responsible for that rule reaches a concentration threshold. When a rule is triggered at a diffusion site, the rule is applied to the current neuron or synapse at that site. .......................... 35

4.1 Different gene types. ............................. 45

5.1 Comparison of the previous methods applied to the Tartarus problem. .............................. 59

5.2 Parameters which are common in the two experiments. ........... 67

6.1 Parameters which are common in both experiments. ............. 81

7.1 Genetic parameters ................................ 100

7.2 Comparison of the best velocities achieved by FGRN and [22] learning algorithm. ............................ 102

7.3 Averaged velocities of the Crawler robot. ..................... 103

7.4 Speeds reached by different types of locomotion ............ 110

7.5 Phase shift (per period) between the neighbor modules for the typical controller of each experiment ................ 111
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>CAM</td>
<td>Cell-Adhesion Molecule</td>
</tr>
<tr>
<td>CPG</td>
<td>Central Pattern Generators</td>
</tr>
<tr>
<td>DNA</td>
<td>DeoxyriboNucleic Acid</td>
</tr>
<tr>
<td>EC</td>
<td>Evolutionary Computation</td>
</tr>
<tr>
<td>FGRN</td>
<td>Fractal Gene Regulatory Network</td>
</tr>
<tr>
<td>FSA</td>
<td>Finite State Automata</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GP</td>
<td>Genetic Programming</td>
</tr>
<tr>
<td>GRN</td>
<td>Gene Regulatory Network</td>
</tr>
<tr>
<td>ISA</td>
<td>If-Statement-Action</td>
</tr>
<tr>
<td>RBN</td>
<td>Random Boolean Networks</td>
</tr>
<tr>
<td>RNA</td>
<td>RiboNucleic Acid</td>
</tr>
<tr>
<td>USSR</td>
<td>Unified Simulator for Self-Reconfigurable Robots</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction
Introduction

Evolutionary Computation (EC) has been extensively used in several engineering fields for several years. EC is implemented in different forms. In the conventional form of EC, phenotypes are encoded directly in the genotype. For example, one may define a genotype as a set of genes each works as a parameter specifying a particular feature in the final phenotype. On the other hand, in the biological systems in nature, this is a process of development which unwraps and decodes genotypes and develops robust and adaptable phenotypes indirectly. Inspired by this natural mechanisms, indirect encoding of evolvable genotypes to phenotypes is becoming a new important area of research in EC in the recent years.

The natural process of development is a lifelong progressive process which can be influenced by environmental signals. This decompression of a genotypes to complex and successful phenotypes is not a trivial process. For example, development of the human brain decompresses the estimated $10^4$ human genes to $10^{14}$ synaptic connections [64]. This process is controlled by an internal network called Gene Regulatory Network (GRN).

A GRN consists of a group of genes that interact with each other in order to control the synthesis of certain products; i.e., proteins. The types and amount of proteins produced by a gene network have a fundamental effect on future expression of genes in the network and consequently the development of the gene network itself. The produced proteins also form the behaviors of the system containing the network. The biochemical processes which are involved in gene expression and regulation are intrinsically stochastic.

The ability of external control of the behavior of the gene networks is an important aspect of the GRNs [21]. Evidence suggests that such a control is possible in nature. An example is the behavior of λ phage [59], a simple virus that chooses one or another mode of growth depending on external signals. These signals caused by different environmental conditions can make completely different phenotypes from a population of genotypically identical developing individuals [76]. Beside differences in environmental signals, other stochastic factors act as sources of versatility in phenotypes arisen from identical genotypes. An interesting biological phenomenon is that a genetically identical population of cells exposed to the same environmental conditions can have phenotypically distinct individuals [60]. This is due to the fact that the biochemical processes which are involved in gene expression and regulation are
intrinsically stochastic. Studies on simple genetic network synthesis have provided experimental information about phenotypic variability and stochasticity of gene expression \[63, 58, 36\].

In the field of computational EC, different models are introduced to imitate GRNs. Various aspects of the biological GRNs are considered in the computational models. The models use different representations and encodings and define various mechanisms in order to extract the encoded network from the represented genotypes. Any encoding and mechanism a GRN model uses and any means of interactions between the genes is employed by the model, a behaving GRN unit is a network of genes regulating each other.

In other word, a computational GRN is made up of a number of nodes which are connected together where each node relates to a gene. Every node is a processing unit which determines the output of the node on the basis of input connections and the related parameters. A GRN may interact with its environment by using a number of input and output nodes.

Although a behaving GRN unit is conceptually similar to recurrent Artificial Neural Networks (ANN) in some ways, these two areas are basically inspired by different types of biological systems and their aspects which attract interest to be observed and investigated by researchers are different. While ANNs \[33\] are inspired by structural and functional aspects of biological neural networks, metabolic networks and the interactions inside a cell in relation to genomic data is the inspiration source of the GRNs. In this way, evolution and developmental behavior during life-time is usually a natural and inherent characteristic of most of the introduced models of GRNs. A typical GRN model usually evolves an encoded substance called genome as a blue-print for developing a behaving GRN during time.

Some models of GRN employ an intermediate substance to play the role of proteins in a biological cell. Proteins emerge from expression of genes and also they influence expression of target genes in the future steps of development. Any expression of a gene increases the level of corresponding protein in the GRN.

Since the computational GRN is still a new area of research, there is no general portrait covering all the aspects and properties of these systems. Also, the potential application areas and capabilities of the systems are not still clear and extensively investigated. This thesis highlights the main features of different GRN models in order to make a uniform perspective. It also tries to give a better understanding of the computational GRNs by investigating the
behavior and capabilities of an example GRN model finding solutions for some different application areas.

A GRN model which is extensively investigated in this dissertation is Fractal Gene Regulatory Network (FGRN). This model utilizes fractals as the intermediate substance of interaction and benefits from the recursive and self-similar nature of fractals in order to make evolvable and complex systems. The model is modified and the capabilities of the system are investigated in different forms and applications.

This thesis is organized in 7 chapters followed by an appendix. In chapter 2 biological cells and their inside interactions are described as the inspiration source of the computational GRNs. Chapter 3 is a literature survey describing various aspects and features of different computational GRNs and briefly reviews several GRN models introduced by researchers in the field. Basic concepts borrowed from biology in FGRN model are indicated and the model is described in details in Chapter 4.

The implemented case studied are described in chapters 5 to 7. Chapter 5 describes a benchmark problem in the field of non-Markovian agent motion planning. FGRN model is modified to incorporate input data in a natural way in order to solve the benchmark problem. The results are compared with other methods and the solution is discussed in detail.

Chapter 6 investigates the evolutionary behavior of an evolving population of GRNs in an interesting situation inspired by social science and swarm intelligence methods. An FGRN system is implemented augmented with the ability of observing the best behavior produced by the individuals of the former evolutionary generations. The evolutionary behavior of the population is described and the role of different types of genes are investigated during generations.

Chapter 7 concerns with the application of FGRN systems in distributed control of modular robots. The model is used in a multi-cellular way. First, the concepts of modular robots are described. Then the system is applied to control the modular robots with different morphologies with no sensor or communication between the modules. The capability of the FGRN system is also evaluated to continue evolution in case of a module failure. In another set of experiments, the implementation of FGRN is extended to a sensor-coupled system. The ability of FGRN is demonstrated in achieving controllers with different levels of complexity in demand of the problem. The solutions are compared with the previous methods and discussed in both cases.
Chapter 2

A Biological Overview


2.1 Introduction

Evolutionary Computation (EC) is inspired by the evolution of biological systems in the real world. Since all the biological organisms are made up of cells as relatively independent building units, studying the internal mechanisms involved in development of the cells might be instructive and inspiring in designing successive evolutionary computational systems.

This chapter presents a very biological viewpoint. It concerns with the construction components, encoding mechanisms, and developmental processes of the biological cells. Cells, genomes, proteins, and developmental procedures in a cell are briefly described in this section. Unless otherwise stated, the material in this section can be found in biological textbooks such as [51] and [19] 1.

![Cell Examples](image)

**Figure 2.1:** a) An example prokaryotic cell; b) An example Eukaryotic cell

2.2 Cell

The biological universe consists of two types of cells: *prokaryotic* and *eukaryotic*. Prokaryotic cells consist of a single closed compartment that is surrounded by the plasma membrane, lacks a defined nucleus, and has a relatively simple internal organization ((Figure 2.1 a)). Bacteria, the most numerous prokaryotes, are single-celled organisms; the cyanobacteria, or blue-green algae, can be unicellular or filamentous chains of cells. Eukaryotic cells, unlike prokaryotics, contain a defined membrane-bound nucleus and extensive internal membranes that enclose other compartments, the organelles (Figure 2.1

---

1 Figures in this section are from wikipedia.org
b). The region of the cell lying between the plasma membrane and the nucleus is the cytoplasm, comprising the cytosol (aqueous phase) and the organelles. Eukaryotes comprise all members of the plant and animal kingdoms, including the fungi, which exist in both multicellular forms (molds) and unicellular forms (yeasts), and the protozoans (primitive-animals), which are exclusively unicellular.

![DNA double helix](image)

**Figure 2.2**: DNA consists of two complementary strands wound around each other to form a double helix

### 2.3 Genome

All of the biological organisms in nature possess a genome. Genome functions as a blueprint of the organism and contains the biological information required for development and maintaining of the living system. In an adult human body, there is approximately $10^{14}$ cells. Every cell has its own copy of the genome, the only exceptions is some cells such as red blood cells, that lack a nucleus in their fully differentiated state and doesn’t have a genome.

Most genomes, including the human genome are made of DNA (deoxyribonucleic acid) but the genome in some organisms such as a few types of viruses is made of RNA (ribonucleic acid).

The information about how, when, and where to produce each kind of protein is carried in the genetic material, which is called DNA. The three-dimensional structure of DNA consists of two long helical strands that are coiled around a common axis, forming a double helix (Figure 2.2). DNA strands are linear unbranched polymers composed of monomers called nucleotides; these often are referred to as bases because their structures contain cyclic organic bases. Four different nucleotides, abbreviated A, T, C, and G, are joined end to end in a DNA strand. Each nucleotide in a DNA polymer is made up of three components.
Human beings share 98 percent of the DNA with chimpanzees and a great percentage of their DNA with other species, including flies and yeast. Only 2 percent of a human genome is exclusively human in character (i.e. occurs only in human beings). Moreover, approximately 0.1 percent of the exclusively human DNA vary from person to person and accounts for much of the genetically caused phenotypic variation we found in our species [62].

Most of the DNA in eukaryotic cells is located in the nucleus, extensively folded into a structures we know as chromosomes. Each chromosome contains a single linear DNA molecule associated with certain proteins. In prokaryotic cells, most or all of the genetic information resides in a single circular DNA molecule about a millimeter in length; this molecule lies, folded back on itself many times, in the central region of the cell. The genome of an organism comprises its entire complement of DNA.

The genetic information carried by DNA resides in its sequence, the linear order of nucleotides along a strand. The information-bearing portion of DNA is divided into discrete functional units, the genes, which typically are 5000 to 100,000 nucleotides long. Most bacteria have a few thousand genes; humans, about 40,000. The genes that carry instructions for making proteins commonly contain two parts: a coding region that specifies the amino acid sequence of a protein and a regulatory region sometimes called promoter region that controls when and in which cells the protein is made.

Figure 2.3: Proteins are greatly different in size, shape, and function

2.4 Protein

Proteins are special type of macromolecules in a cell and give the cell structure and perform almost all the activities of the cell. Proteins are encoded by the genes and are greatly different in size, shape, and function (See Figure 2.3).
Some proteins function as catalyzers of chemical reactions for intracellular and extracellular activities. These proteins are called enzymes. Other proteins can be grouped into other functional classes such as:

- **structural proteins**, which provide structural rigidity to the cell;
- **transport proteins**, which control the flow of materials across cellular membranes;
- **regulatory proteins**, which act as switches to control protein activity and gene function;
- **signaling proteins**, including cell-surface receptors and other proteins that transmit external signals to the cell interior and helps sensing and reacting to the environment of the cell;
- **motor proteins**, which cause motion.

The collection of all the proteins present in a cell at a particular time is called proteome. Proteome of the yeast *Saccharomyces cerevisiae* consists of about 6000 different proteins. However, the human proteome is only about five times as large, comprising about 32,000 different proteins.

There are very few differences between the proteomes of different types of mammalian cells. It suggests that most of the proteins are housekeeping proteins that perform general biochemical activities that are similar in all cells. The proteins that provide the cell with its specialized function are often quite rare, although there are exceptions such as the vast amounts of hemoglobin that are present only in red blood cells (Alberts et al., 1994).

Scientists classify many proteins in an organism’s proteome by comparing protein sequences and structures with proteins of known function. The function of a protein whose structure has not been determined can often be inferred from its interactions with other proteins, from the effects resulting from genetically mutating it, from the biochemistry of the complex to which it belongs, or from all three.

![Figure 2.4: Amino acids](image-url)
The monomeric building blocks of proteins are *amino acids*. All the variety of proteins are built from only 20 different amino acids (Figure 2.4) and the resulting polypeptides (proteins) are rarely more than 2000 units in length. Amino acids have a common characteristic structure. They are all consist of a central $\alpha$ carbon atom ($C_\alpha$) bonded to four different chemical groups and by differences in a chemical residue called the side-chains which varies in size, shape, charge, hydrophobicity, and reactivity, they have different characteristics.

Although constructed by the polymerization of only 20 different amino acids into linear chains, proteins perform incredibly diverse tasks. A protein chain folds into a unique shape that is stabilized by noncovalent interactions between regions in the linear sequence of amino acids. Only when a protein is in its correct three-dimensional structure, or conformation, it is able to function efficiently. A key concept in understanding how proteins work is that function is derived from three-dimensional structure, and three-dimensional structure is specified by amino acid sequence.

A protein’s structure is often referred from four different hierarchical levels (Figure 2.5):

- **Primary structure** of a protein is simply the linear arrangement, or sequence, of the amino acid residues that compose it. Many terms are used to denote the chains formed by the polymerization of amino acids. A short chain of amino acids linked by peptide bonds and having a defined sequence is called a peptide; longer chains are referred to as polypeptides. Peptides generally contain fewer than 20 to 30 amino acid residues, whereas polypeptides contain as many as 4000 residues. We generally reserve the term protein for a polypeptide (or for a complex of polypeptides) that has a well-defined three-dimensional structure.

- **Secondary structures** are the various spatial arrangements resulting from the folding of localized parts of a polypeptide chain which are stabilized by hydrogen bonds. Certain combinations of secondary structures give rise to different motifs, which are found in a variety of proteins and are often associated with specific functions.

- **Tertiary structure** is the overall conformation of a polypeptide chain, that is, the three-dimensional arrangement of all its amino acid residues. Tertiary structure is primarily stabilized by hydrophobic interactions between the nonpolar side chains, hydrogen bonds between polar side
chains, and peptide bonds. These stabilizing forces hold elements of secondary structure compactly together. Because the stabilizing interactions are weak, however, the tertiary structure of a protein is not rigidly fixed but undergoes continual and minute fluctuation. This variation in structure has important consequences in the function and regulation of proteins. Large proteins often contain distinct domains, independently folded regions of tertiary structure with characteristic structural or functional properties or both. The incorporation of domains as modules in different proteins in the course of evolution has generated diversity in protein structure and function.

- **Quaternary structure** encompasses the number and organization of several protein molecules (polypeptide chains), usually called protein subunits in this context, which function as a single protein complex.

The sequence of a protein determines its three-dimensional structure, which determines its function. In short, function derives from structure; structure derives from sequence.

A polypeptide chain is synthesized by a complex process called translation in which the assembly of amino acids in a particular sequence is dictated by messenger RNA (mRNA). The cell promotes the proper folding of a nascent polypeptide chain and, in many cases, modifies residues or cleaves the polypeptide backbone to generate the final protein. In addition, the cell has error-checking processes that eliminate incorrectly synthesized or folded proteins.

The life span of intracellular proteins varies from as short as a few minutes for mitotic cyclins, which help regulate passage through mitosis, to as long as the age of an organism for proteins in the lens of the eye. Eukaryotic cells have several intracellular proteolytic pathways for degrading misfolded or denatured proteins, normal proteins whose concentration must be decreased, and extracellular proteins taken up by the cell.
2.5 Development of an organism

Every human being begins as a zygote, which houses all the necessary instructions for building the human body containing about $10^{14}$ cells. Development begins with the fertilized egg cell dividing into two, four, then eight cells, forming the very early embryo. Continued cell proliferation and then differentiation into distinct cell types gives rise to every tissue in the body. One initial cell,
the fertilized egg (zygote), generates hundreds of different kinds of cells that differ in contents, shape, size, color, mobility, and surface composition.

Making different kinds of cells (muscle, skin, bone, neuron, blood cells) is not enough to produce the human body. The cells must be properly arranged and organized into tissues, organs, and appendages. Our two hands have the same kinds of cells, yet their different arrangements (in a mirror image) are critical for function. In addition, many cells exhibit distinct functional and/or structural asymmetries, a property often called polarity. From such polarized cells arise asymmetric, polarized tissues such as the lining of the intestines and structures like hands and hearts.

Cells use two processes in series to convert the coded information in DNA into proteins (Figure 2.6). In the first, called transcription, the coding region of a gene is copied into a single-stranded ribonucleic acid (RNA) version of the double-stranded DNA. A large enzyme, RNA polymerase, catalyzes the linkage of nucleotides into a RNA chain using DNA as a template. In eukaryotic cells, the initial RNA product is processed into a smaller messenger RNA (mRNA) molecule, which moves to the cytoplasm. Here the ribosome, an enormously complex molecular machine composed of both RNA and protein, carries out the second process, called translation. During translation, the ribosome assembles and links together amino acids in the precise order dictated by the mRNA sequence according to the nearly universal genetic code.

All organisms have ways to control when and where their genes can be transcribed. For instance, nearly all the cells in our bodies contain the full set of human genes, but in each cell type only some of these genes are active, or turned on, and used to make proteins. That’s why liver cells produce some proteins that are not produced by kidney cells, and vice versa. Moreover, many cells can respond to external signals or changes in external conditions by turning specific genes on or off, thereby adapting their repertoire of proteins to meet current needs. Such control of gene activity depends on DNA-binding proteins called transcription factors, which bind to DNA and act as switches, either activating or repressing transcription of particular genes. Transcription factors are shaped so precisely that they are able to bind preferentially to the promoter regions of just a few genes out of the thousands present in a cell’s DNA.

The coded information in DNA is converted into the amino acid sequences of proteins by a multistep process (See Figure 2.6)as follows:

- Step 1: Transcription factors bind to the regulatory regions of the specific
Figure 2.6: The coded information in DNA is converted into the amino acid sequences of proteins by a multistep process.

Figure 2.7: DNA replication.

genes they control and activate them.

• Step 2: Following assembly of a multiprotein initiation complex bound to the DNA, RNA polymerase begins transcription of an activated gene at a specific location, the start site. The polymerase moves along the DNA linking nucleotides into a single-stranded pre mRNA transcript using one of the DNA strands as a template.

• Step 3: The transcript is processed to remove noncoding sequences.

• Step 4: In a eukaryotic cell, the mature messenger RNA (mRNA) moves to the cytoplasm, where it is bound by ribosomes that read its sequence and assemble a protein by chemically linking amino acids into a linear chain.
Every time a cell divides, a large multiprotein replication machine, the replisome, separates the two strands of doublehelical DNA in the chromosomes and uses each strand as a template to assemble nucleotides into a new complementary strand (Figure 2.7). The outcome is a pair of double helices, each identical to the original.

Mistakes occasionally do occur spontaneously during DNA replication, causing changes in the sequence of nucleotides. Such changes, or mutations, also can arise from radiation that causes damage to the nucleotide chain or from chemical poisons that lead to errors during the DNA-copying process. Mutations come in various forms: a simple swap of one nucleotide for another; the deletion, insertion, or inversion of one to millions of nucleotides in the DNA of one chromosome; and translocation of a stretch of DNA from one chromosome to another.

Sequencing of the human genome has shown that a very large proportion of our DNA does not code for any RNA or have any discernible regulatory function. Mutations in these regions usually produce no immediate effects; good or bad. However, such "indifferent" mutations in nonfunctional DNA may have been a major player in evolution, leading to creation of new genes or new regulatory sequences for controlling already existing genes. For instance, since binding sites for transcription factors typically are only 10 to 12 nucleotides long, a few single nucleotide mutations might convert a nonfunctional bit of DNA into a functional protein-binding regulatory site.

Much of the nonessential DNA in both eukaryotes and prokaryotes consists of highly repeated sequences that can move from one place in the genome to another. These mobile DNA elements can jump (transpose) into genes, most commonly damaging but sometimes activating them. Jumping generally occurs rarely enough to avoid endangering the host organism. Mobile elements, which were discovered first in plants, are responsible for leaf color variegation and the diverse beautiful color patterns of Indian corn kernels. By jumping in and out of genes that control pigmentation as plant development progresses, the mobile elements give rise to elaborate colored patterns. Mobile elements were later found in bacteria in which they often carry and, unfortunately, disseminate genes for antibiotic resistance. Now we understand that mobile elements have multiplied and slowly accumulated in genomes over evolutionary time, becoming a universal property of genomes in present day organisms. They account for an astounding 45 percent of the human genome. Some of our own mobile DNA elements are copies (often highly mutated and damaged)
of genomes from viruses that spend part of their life cycle as DNA segments inserted into host–cell DNA. Thus we carry in our chromosomes the genetic residues of infections acquired by our ancestors. Once viewed only as molecular parasites, mobile DNA elements are now thought to have contributed significantly to the evolution of higher organisms.

2.5.1 Communication

The simplest multicellular animals are single cells embedded in a jelly of proteins and polysaccharides called the extracellular matrix. Cells themselves produce and secrete these materials, thus creating their own immediate environment.

The cells in animal tissues are "glued" together by cell-adhesion molecules (CAMs) embedded in their surface membranes. Some CAMs bind cells to one another; other types bind cells to the extracellular matrix, forming a cohesive unit.

The cytosols of adjacent animal or plant cells often are connected by functionally similar but structurally different "bridges". These structures allow cells to exchange small molecules including nutrients and signals, facilitating coordinated functioning of the cells in a tissue.

A living cell continuously monitors its surroundings and adjusts its own activities and composition accordingly. Cells also communicate by deliberately sending signals that can be received and interpreted by other cells. Such signals are common not only within an individual organism, but also between organisms. Cells possess numerous receptor proteins for detecting signals and elaborate pathways for transmitting them within the cell to evoke a response. At any time, a cell may be able to sense only some of the signals around it, and how a cell responds to a signal may change with time. In some cases, receiving one signal primes a cell to respond to a subsequent different signal in a particular way. Both changes in the environment and signals received from other cells represent external information that cells must process.

The ability of cells to send and respond to signals is crucial to development. Many developmentally important signals are secreted proteins produced by specific cells at specific times and places in a developing organism. Often a receiving cell integrates multiple signals in deciding how to behave, for example, to differentiate into a particular tissue type, to extend a process, to die, to send back a confirming signal, or to migrate.
In addition to modulating the activities of existing proteins, cells often respond to changing circumstances and to signals from other cells by altering the amount or types of proteins they contain. Gene expression, the overall process of selectively reading and using genetic information, is commonly controlled at the level of transcription, the first step in the production of proteins.

### 2.5.2 Regulation activities

Control of gene activity in cells is not trivial [1, 20]. It usually involves dual positive/negative control of gene expression which is performed by a balance between the actions of transcriptional activator proteins and repressor proteins. Binding of activators to specific DNA promoter sequences called enhancers turns on transcription, and binding of repressors to other promoter sequences called silencers turns off transcription. Activator and repressor proteins might be produced based on either the internal activities of the cell or a product of communications with other cells or environment.

Considering the fact that the transcriptional activator/repressor proteins are result of expression of some genes to proteins, a network of gene regulation is produced which is called Gene Regulatory Network (GRN). Development of the body plan is controlled by the large networks of regulatory genes. The complex control systems underlying development have probably been evolving for more than a billion years. They regulate the expression of thousands of genes in any given developmental process. They are essentially hardwired genomic regulatory codes, the role of which is to specify the sets of genes that must be expressed in specific spatial and temporal patterns. In physical terms, these control systems consist of many thousands of modular DNA sequences. Each such module receives and integrates multiple inputs, in the form of regulatory proteins (activators and repressors) that recognize specific sequences within them. The end result is the precise transcriptional control of the associated genes. Some regulatory modules control the activities of the genes encoding regulatory proteins. Gene regulatory networks explicitly represent the causality of developmental processes. They explain exactly how genomic sequence encodes the regulation of expression of the sets of genes that progressively generate developmental patterns and execute the construction of multiple states of differentiation [24].
Chapter 3

Literature Survey
3.1 Introduction

In order to apply an indirect mapping from genotypes to phenotypes, different approaches have been tried by researchers. Some approaches evolve a grammar which is repeatedly applied to the phenotype [35, 39, 47, 52]. In some other approaches inspired by natural cells and DNA system, GRNs are evolved with their own metabolism and chemistry.

A GRN is made up of a number of nodes which are connected together as a network. Each node of the network represents a gene. Thus, every connection directed from a node to another one represents the influence of the first gene to the other one (Figure 3.1).

A typical GRN model has an encoding genome which is a set of genes and functions as a blueprint to make the GRN system. The structure of a genome and the representation of each gene specifies the structure of the network encoded by the genome. These features are different from model to model and require different processes for unwrapping and decoding the genome in order to make the functional GRN system.

The activity pattern of the nodes (genes) and the generated output patterns of the system during development is a main subject of interest in the GRN systems. Activation patterns of GRNs can be used to develop structures and functionalities during time. For example they can develop patterns and morphologies, create multicellular systems, and make neural networks. Since a GRN system can be influenced by the outside environment through input signals, instructions and feedbacks from environment influence development during the process.

Many researchers have introduced their own GRN models. These models are different and similar in various ways and can be discussed from different aspects. In the following sections, first the various interesting aspects of the computational GRN models are discussed and then several models employing different approaches are described. Note that the FGRN model which is the main focus of this dissertation will be described very detailed in the following separate chapter.
3.2 Various aspects of GRNs in different computational models

Computational GRN models define various mechanisms to imitate different aspects of the natural GRNs. Since the real mechanisms of the biological systems are very complicated and are not fully understood, researchers need to make selections and simplifications depending on the limitations and purposes. They need to devise representations and mechanisms with approximately similar properties as the selected biological mechanisms. One of the influential parameters in making the choice of the most interesting properties of the real systems is the purpose of defining the artificial model. For example, some models are designed basically for studying the properties and behaviors of the network model. Other models are designed having special applications in mind.

GRN models typically include an encoding genome which is a set of genes and functions as a blue-print to make the GRN system. The structure of a genome and the representation of each gene specifies the structure of the network encoded by the genome. These features vary from model to model and require different processes for unwrapping and decoding the genome in order to make the functional GRN system.

In some models of GRNs, the network structure is encoded in a straightforward way such that one can simply draw the encoded network by looking at the genome. In other models extracting the encoded network needs more effort and more complex process and is usually not practical to be done manually.

One of the main differences between the various GRN models is the way
they encode the connections between the nodes. In some models, networks are based on direct encoding of the connections. The representation of genes in these models directly specifies which genes influence which ones and indicates the connections between the nodes of the network. For example, a model may use an identification number to indicate each gene. On the other hand, it may assign a set of numbers to every gene in order to identify the genes which influence that particular one.

In other models, the connection between the genes is indirectly encoded in the representation of the genome. For example, each gene might be represented by a string of digits and the system uses a pattern matching technique functioning on the strings in order to specifying the genes and extracting the connections between them. More complicated mechanisms are defined in some models in order to make complex evolvable systems. They may include a mechanism for translating a particular part of a gene to an intermediate substance - namely protein- and a mechanism to calculate the result of interaction between the intermediate substances and a gene.

Another aspect which varies in different GRN models is the transfer function of the network’s nodes or in other word, the gene’s products. In some models, genes function as boolean nodes in the network that means a gene behaves as an on/off switch when it is activated or inactivated. This is in contrast with the models which implement concentration levels of gene products. In these models, an activated node in the network makes some non-boolean value as the outputs. This value can be added to the current value of a variable representing the concentration level of that particular gene product in the system and might be a subject of post-processing operations such as degradation in order to assimilate the natural biological GRNs. In the same way, activation/inactivation of a gene might be regulated considering the concentration levels of the genes which influence it.

Another aspect of the GRN models is multicellularity. A multicellular GRN system is a system consisting of several GRN units called cells where all the units are created based on an identical genome and all of the units together function as a whole. Some of the GRN models are designed particularly to be used to develop a multicellular GRN system. In a developing multicellular GRN system, cells might be added, removed, or displaced based on the model. These changes may lead to substantial variations in the network structure of the whole system during its life-time. Division mechanisms are usually defined in these models in order to make new GRN cells. In the same way, com-
Communication mechanisms between the cells specify the inter-GRN connections. Maternal substrates and the gradient mechanisms in the cell’s environment might be also defined in these models to make variations in internal state of the different cells and/or influence the development of the organism.

Other GRN models are basically designed as a single-cell system. These models can be also implemented in a multicellular way although sometimes not very conveniently because of their main design.

One of the features which arises in a multicellular system is a definition for communication mechanisms. Whether a multicellular system develops from one or more initial cells or the final number of cells are created from the first step, some communication mechanisms are required in order to establish interactions between the cells which maintain individual GRNs. Maternal substrates and the gradient mechanisms in the cell’s environment might be also defined in some models to make variations in internal state of the different cells and/or to influence the development of the system. These mechanisms can be defined differently in each model.
3.3 Kauffman’s model

One of the first models of GRN are Random Boolean Networks (RBNs). RBNs are proposed by Kauffman in 1969 [44] as the model of GRNs with direct-connection encoding (Figure 3.1). These models are also called NK models or Kauffman models. An RBN system is a system made up of N binary-state nodes where each node indicates a gene. To each node, one of the possible Boolean functions and a number of inputs from other node are assigned. The inputs are selected randomly among all the N nodes of the system. In every time step, each node is activated or inactivated by applying the Boolean function to the state of the inputs from the previous time step. Thus the state space size of the network is $2^N$. This model is deterministic in its classical form. The model has not been used in its initial form to solve any practical problem. It is mainly designed to study the dynamics of GRN systems in general as a ubiquitous phenomenon in complex systems. It is widely studied in the context of network theory as a complex network and the properties of the network are investigated; for example, the state space attractors of the network (See Figure 3.2 for an example), activity patterns, and the dynamic effects of perturbations in the states have been studied thoroughly ([45]).

![Figure 3.2: State space of an RBN with 10 nodes (self connections are not shown).](image-url)
3.4 Dellaert’s model

Authors in [26] introduced a multicellular GRN based on RBN which develops a neural network. The system is implemented for a line-following agent.

The system starts with one cell. Cells divide on the basis of their RBN state-vector and develop a multicellular RBN system. All of the cells contain an identical copy of the same RBN genome but the state of the RBN might vary from cell to cell. The appropriate boolean functions of the nodes are specified in the genome and are evolved by genetic algorithm. The genome also specifies which nodes are connected to which ones in the network.

Each node maintains a state-vector. Since the outputs of the nodes of an RBN are boolean (activated/disactivated), state-vector of a cell is a vector of bits.

Some of the nodes are interpreted as output nodes and their output bit represents a special instruction for the cell. For example, if the output bit of division-node is 1, the cell is divided. In order to implement symmetry-breaking, a node in one of the children in the first cell-division flips the state. The system initially starts with a cell having the division bit set to 1 manually in order to encourage the process of development.

Communication between the cells is also implemented by receiving a set of inputs from neighboring cells which perturbs state of the cell’s RBN. Cells can receive some information about their placement as well. For example, the cells in the perimeters of the system or the cells in the midline have the corresponding bits set to 1 in the cell’s vector of state.

Some nodes of the RBN network are interpreted as the neural network development nodes. The outputs of these node are instructions for neural network development. The neural network is developed on top of the cells. The related nodes of the RBN determine the neural type of the cell. These nodes specify whether the cell will send out an axon, it is a target for innervation, it is inhibitory or excitatory connection if innervation happens in the cell, or it is a sensor/actuator, and so on.

Development of a neural network is simple. For example, each cell that has the ’axon bit’ set will innervate all the cells with the ’target bit’ set within its range. The range at which cells innervate each other and the connection weights are evolved together with the genome, and are identical for all the cells. Thus, the RBN develops a multicellular system containing the RBNs with the same encoding.
Cell-division which leads to development of the multicellular system continues for a pre-specified maximum number of steps. After development of the morphological structure of the system, RBNs develop the neural network by specifying the type of the cell in relation to the neural network and the sign of the connections for the cells. Evolution specifies the innervation range and the weight factors of the neural network along with the RBN structure and the boolean functions of its nodes in order to obtain the desired behavior.
3.5 Reil’s model

[61] introduces a GRN model which is basically different from RBN in the way of describing connections between the genes. This model defines a pattern matching (template matching) mechanism for specifying which nodes are connected to which ones that represents the genes influencing each other.

A string of randomly generated digits represents the genome in this model and contains all the information present. A genome contains both coding and non-coding parts. After creating a genome, it is searched for coding parts to make the network. The rest of the genome will be discarded.

The genome is searched for a pre-specified sequence of digits. N consecutive digits following this sequence are considered a gene. The model defines a substrate called protein which is product of a gene. A protein is also a sequence of digits and is generated based on the gene’s string. A protein can be defined as a string of digits where each value in the protein is the corresponding value in the gene incremented by one. So that the gene sequence ‘221133’ becomes a protein ‘332244’. The genome is then searched for occurrence of this protein sequence and finding the sequence is considered a promoter part for the first gene which follows it. That means all of the first genes after all the occurrences of the sequence ‘332244’ in a genome are influenced by the gene ‘221133’. Using this method, all the connections between the genes are identified and the network is created. Each gene in the network is influenced by a number of genes.

The influence of a gene can function as an enhancer or inhibitor which is defined based on its string of digits. For example it can be defined such that if the last digit of a gene is an even number, the gene functions as inhibitor for the other genes, otherwise it is an enhancer. This model doesn’t take concentrations into account and the boolean activation function of the genes are simple. If one of the inhibitors is activated for a gene it will be inactivated. Otherwise, the presence of single enhancer suffices to activate the gene.

The main goal of this model is to study the pattern of on/off genes over time. In this regard, the dynamic properties of the gene’s activities are identified.
3.6 Jacobi’s model

One of the GRN models which employs concentration levels of gene products is [41]. This model is mainly implemented to develop a neural network in order to control a robot. The model is designed as a multicellular GRN system where the cells divide and move in response to the interactions in the GRNs. After completion of the growth of the multicellular organism, connections between the nodes of the neural network grow.

The genome in this model is a string of digits (or characters). After creating a genome, it will be searched for the genes in order to specify the GRN. All the genes in the genome are of the same length and start with a specific string which identifies the gene. A gene consists of the start indicator, a link template, a threshold, a protein type, and a protein specification (Figure 3.3). Each gene is responsible to produce a particular protein and is regulated by a certain combination of proteins. The amount of each protein present in the cell has a value in the range of \([0, 1.0]\).

One part of each gene specifies a protein. The protein is the string encoded in the gene where both ends of the string are connected together and makes a circle. In order to determine if a protein participates in regulation of a gene, this circular string rotates and each time a part of it is examined to be matched against the link template of the gene. If the matching degree is more than the threshold value of the gene, the protein influences the gene. The contribution this protein makes to the regulation of the gene is then calculated from matching the diametrically opposite side of the protein circle with an arbitrary template which is specified for this purpose (Figure 3.4). Similar process is performed to calculate the influence of one protein to another when two proteins interact which is the case in some particular types of proteins.

Proteins are classified in six types, each play a specific role in the organism.
Figure 3.4: A protein has been rotated until the match between the link template on the gene and the corresponding protein segment goes over a certain threshold. The contribution this protein makes to the regulation of the gene is then calculated from matching the diametrically opposite side of the protein with an arbitrary fixed template.

- **Signal** proteins diffuse out of one cell and into another without any functionality in the source cell. They also may be used as maternal factors which are emitted from dependent sources in the environment of the organism. These proteins either influence the internal dynamics of the cell they are entered or moving the cell towards or away from the protein source interacting the mover proteins.

- **Mover** proteins control the way in which a cell moves. Each mover protein is influenced by a number of signal proteins which match it more than a certain threshold.

- **Splitter** proteins determine when and how a cell should split (divide). In each time-step, a weighted sum of the maximum matching degrees between splitters rotating around a pre-specified templates related to splitting behavior are calculated and if the value exceeded a threshold the cell splits to two cells. There are two pre-specified templates which splitters match against them and each of them relates to the way of splitting regarding the direction.
• **Differentiator** proteins work in a similar way as splitters. A weighted sum of maximum matching between the differentiators and a pre-specified template is calculated and added to a variable keeping the differentiator concentration. When the total concentration exceeds a threshold, it is considered in differentiation state. In this case, the cell stops in its current situation and waits for the other cells to get to differentiation state. When the whole system reached to the differentiation state, the organism starts being translated to a neural network.

• **Dendrite** proteins control the way in which neural network dendrites grow. The concentration of a dendrite protein is investigated after differentiation. The number of dendrites that will grow from a cell is the number of dendritic proteins whose concentration lies above a threshold associated with that protein. Dendrites also respond to a set of signal proteins which matches against them.

• **Threshold** proteins are responsible for setting the thresholds on the units of the neural network.

Initially a single cell is placed in an environment. The environment contains a number of signal protein sources as references for development of the organism.

At each time step the concentrations of every signal protein in the local environment of each cell is calculated. New inputs for all nodes in each GRN are calculated from the external signal proteins and the recurrent connections, and new outputs are calculated. All internal variables are then updated.

Forces on each cell are calculated based on the mover proteins and the cell moves in the direction of the resultant forces.

If any internal variable for splitting or for differentiation has gone over its threshold, then the appropriate action takes place. When a cell reaches to differentiation state, the concentrations within the cell and the levels of signal proteins output from the cell become fixed. After differentiation of all the cells, the neural network develops based on the dendrites and threshold proteins of the cells. The system is evolved for robot controllers for both corridor following and object avoidance tasks.
3.7 Eggenberger’s model

[28] introduces a model to develop multi-cellular organisms. This model introduces communication mechanisms and relies on diffusion of proteins between the cells and between environment and the cells. It uses the concept of morphogenes as maternal proteins located in the environment of the organism. The model also defines cellular movement to determine connections between the cells of the organism.

Genome in this model is a string of digits which makes a number of consecutive parts. Each part consists of one or more regulatory sites and one or more coding regions which are controlled by those regulatory sites. Every region is identified by a pre-specified code.

![Figure 3.5: Different regulation schemes: a) intracellular regulation. b) intercellular regulation. c) The intercellular communication through on a receptor which depends on the affinity between the receptor and the regulatory protein](image)

Coding regions produce proteins or determine behavior for the cell. Their product is belonged to one of the different types:

- **Regulatory proteins (called transcription factors)** which are used to regulate the activity of the genes.

- **Cell Adhesion Molecules (CAM)** which are used to connect two cells to each other if both cells have CAMs with enough affinity (similarity). CAMs can function as springs (or similar compliments) for keeping the
cells close to each other, or they can be defined as connections between the nodes of a neural network. Neighboring cells are searched in a range which is specified in the gene in order to find a matching CAM.

- **Receptor proteins** are used to regulate the communication between the cells.

- **Artificial functions** which determine an action for the cell. For example, division, death, migration and search.

Regulatory proteins have one of the following functionalities (See Figure 3.5):

- Only influence genes inside their containing cell.
- Diffuse to the nearby cells and capable of influencing the genes there.
- Diffuse to the nearby cells only if there is enough affinity between them and the receptor proteins at the surface of those cells.

The degree of activation (or inactivation) of each coding region is calculated based on the affinity between the regulatory proteins existing in the cell and the related regulatory sites, and also the concentration level of the regulatory proteins. The affinity is calculated using a defined affinity function.

The gene regulation and cell–cell signaling dynamics is defined by differential equations that take into account the interaction between gene products and their diffusion in the developing structure. The physics of cell–cell interaction is modeled using a physical simulator that implements Newton’s laws and provides the possibility of defining elastic and viscous forces acting on cells. The model is capable of evolving a developmental process that creates aggregates of cells displaying morphologies similar to those observed in the early stages of vertebrate embryo development. The system is also used to successfully evolve functional structures such as optical lenses and neural networks for robot control. [34, 32, 31, 30, 29, 28]
3.8 Bongard’s model

The model introduced in [17, 16, 18] is implemented to evolve both morphology and neural structure of a multicellular organism as a simulated agent consists of a number of units. Each agent starts development from a single unit and develop to a multi-unit agent. Every unit has six diffusion sites located along the six line segments between the center of the unit’s sphere and the surface pointing north, south, west, east, up and down.

Each diffusion site contains zero or more proteins and sensor, motor and internal neurons. Newly generated units attach to the unit in the position of these sites (See Figure 3.6). Proteins diffuse from each diffusion site to its neighboring sites within the unit or to the neighboring unit connected to that site. In the same way, the neurons at a diffusion site may be connected to other neurons at the same diffusion site, or another diffusion site within the same unit or in other units.

A genome in this model is generated as a random sequence of floating point values range between 0.00 and 1.00. The genome is then searched for values in specific range (i.e. between 0 and 0.1) Such a value indicates the start of the genes along the genome.

Figure 3.6 shows a part of an example genome directing the growth of a 3-unit agent. The seven floating-point values following a start indicator of a gene, supply the parameter values for the gene. They indicate the protein which regulates expression of the gene, whether it is activated or inactivated in presence of the regulatory protein, the range of required concentration for activation/inactivation of the gene, the concentration level of the expressed protein, and the diffusion site the protein is placed in if the gene is expressed.

There are 24 different types of proteins. Two proteins affect the growth of the radius of the unit in which they diffuse. 17 other proteins affect the growth of the agent’s neural network. Five other proteins are regulatory and may affect expression of genes instead of having any phenotypic effect.

Each agent begins development as a single unit. The initial single unit is injected with a small amount of some proteins at diffusion site 1. During the initial time steps of development, the gene product diffuses to the neighboring four diffusion sites, and thence into the diffusion site diametrically opposite to site 1. In this way diffusion gradient is established within the unit, analogous to the establishment of a gradient of maternal proteins in fruit flies in nature, which leads to the determination of the primary body axis within these
organisms.

The growth of radius of a unit is controlled by two proteins concentrations. If the radius of the unit reaches twice of its original radius, a split event occurs. The radius of the parent unit is halved, the gene product diffusion site with the maximum growth enhancer protein is located, and a new unit is attached to the parent unit at this position. Half of the amounts of all the proteins at this diffusion site are moved to the neighboring diffusion site in the new unit.

A newly-created unit is attached to its parent unit in one of six possible directions, and is connected by a rigid connector that maintains a constant distance between the units. The parent unit is fixed to the rigid connector but the new unit is attached to the rigid connector by a one degree of freedom rotational joint.

In addition to the morphology of the agent, neural structure may grow within the developing agent. Different types of neurons are connected to each other by synapses. During development, neurons can migrate between different units. If the neurons are connected by a synapse, this may lead to signal propagation between neurons in distant units.

Cellular encoding is employed to develop the neural network in the same time as the development of the morphology. Cellular encoding is a method
for evolving both the architecture and synaptic weights of a neural network by starting with a simple neural network, and iteratively or recursively applying a set of graph rewrite rules to transform the simple network into a more complex network. In this GRN model, high concentrations of certain gene products triggers graph rewriting rules that modify or increase the amount of neural structure in a unit. In this way, both morphology and neural structure can change together during the growth phase. The description of the cellular encoding operations related to the proteins in this model is shown in Table 3.1 Once development is complete, the neural network that has grown within the agent is activated in order to make the system operational.
Table 3.1: Phenotypic effect of neural development gene products

The rewrite rules are triggered when the gene product responsible for that rule reaches a concentration threshold. When a rule is triggered at a diffusion site, the rule is applied to the current neuron or synapse at that site.

<table>
<thead>
<tr>
<th>Protein</th>
<th>Cellular Encoding Operation Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Split the current neuron into two neurons. Move the output synapses of the original neuron to the new neuron. Connect the original neuron to the new neuron with a synapse of positive weight.</td>
</tr>
<tr>
<td>1</td>
<td>Split the current neuron into two neurons. Copy the input and output synapses to the new neuron. Connect the two neurons to each other using two synapses of positive weight.</td>
</tr>
<tr>
<td>2</td>
<td>Move the current neuron to the previous diffusion site.</td>
</tr>
<tr>
<td>3</td>
<td>Move the current neuron to the next diffusion site.</td>
</tr>
<tr>
<td>4</td>
<td>Move the head of the current synapse to the current neuron.</td>
</tr>
<tr>
<td>5</td>
<td>Move the tail of the current synapse to the current neuron.</td>
</tr>
<tr>
<td>6/7</td>
<td>Increment/decrement the weight of the current synapse by 0.01.</td>
</tr>
<tr>
<td>8</td>
<td>Duplicate the current synapse.</td>
</tr>
<tr>
<td>9</td>
<td>Delete the current neuron, including any ingoing and outgoing synapses.</td>
</tr>
<tr>
<td>10</td>
<td>Delete the current synapse.</td>
</tr>
<tr>
<td>11/12</td>
<td>Move the neuron pointer to the next/previous neuron at the current diffusion site.</td>
</tr>
<tr>
<td>13/14</td>
<td>Move the synapse pointer to the next/previous synapse at the current diffusion site.</td>
</tr>
<tr>
<td>15/16</td>
<td>Change the type of the current neuron.</td>
</tr>
</tbody>
</table>
3.9 Banzhaf’s model

Another GRN model which uses pattern matching and employs concentration levels is introduced in [8, 7].

A sequence of bits represents a genome in this model and contains a number of genes. Beginning of a gene is specified by a particular sequence of bits. By iterating the bit sequence of the genome, genes are discovered. After the particular sequence, two specific sites are located: an enhancer site and an inhibitor site. Proteins can attach to these sites and enhance or inhibit the protein production of the gene (see Figure 3.7).

Each Protein in this system is also a sequence of bits in the same size of enhancer/inhibitor sites. The bit-sequence of a protein is produced by iterating bits of the related gene information sequence and performing a many-to-one mapping using a majority rule (Figure 3.7).

The matching degree between a protein and an enhancer/inhibitor site is determined by counting the number of matched bits achieved by performing an XOR operation between the protein’s bit-sequence and the site’s bit-sequence.

The matching degree is calculated for each protein and the enhancer/inhibitor sites of all the genes. This value specifies the degree of attachment of the protein to the gene’s site and it’s influence in the protein production rate of the gene is exponential. Both the matching degree and the current amount (concentration level) of the attached protein participate in the protein production rate of the gene.

The enhancing and inhibitory signals for a gene are calculated on the basis of the matching degree and concentration level of each protein as follows:
\[ e_i = \frac{1}{N} \sum_j c_j e^{\beta(u_j^+ - \frac{u_{\text{mas}}}{u_{\text{mas}}})} \]  

(3.1)

\[ in_i = \frac{1}{N} \sum_j c_j e^{\beta(u_j^- - \frac{u_{\text{mas}}}{u_{\text{mas}}})} \]  

(3.2)

where \( c_j \) is the concentration level of gene \( j \), \( u_j \) is the matching degree of protein \( j \), and \( u_{\text{mas}} \) is the maximum achievable matching degree.

The amount of change in the protein concentration is calculated as follows:

\[ \frac{dc_i}{dt} = \delta(e_i - in_i)c_i - \Phi \]  

(3.3)

where \( \delta \) is a positive scaling factor, \( \Phi \) is a term that proportionally scales protein production, ensuring that \( \sum_i c_i = 1 \) which results in competition between binding sites for proteins [55].

A GRN in this model is considered a fully connected network where all the nodes influence each other in different levels. The output from each node is the concentration value of the related protein at the current time step. (Figure 3.8).

As it is introduced in an extended version of this model [55], some proteins can be defined as output proteins. These proteins have no influence in regulating protein production and their concentration level is only used as the outputs of the system. In the same way, some proteins can be defined as input proteins which are not under control of regulatory network and their concentration level is specified by the environment as the received input values.

This GRN model is basically designed as a single-cell GRN in order to study the properties and behavior of the network and activation/concentration patterns. The properties are studied with different initializations of the genome.
for example, initialization by either random bit sequence or looping through successive stages of duplication and divergent mutation. The model has been used for some applications such as function approximation \[50, 27\] and pole balancing \[55\].
3.10 Taylor’s model

A GRN model introduced in [67] is used as controller of a group of underwater robots. This model is not designed to develop a multicellular organism but initially there is a GRN in each robot as a cell, and protein diffusion is employed as a method for communication between the GRNs.

Connections are specified directly using protein identifier numbers which connect the nodes to each other. Genome in this model is a randomly generated sequence of digits (in base-4). After creating a genome, it is searched to identify specific patterns of digits. There are three different patterns which specify the coding regions, enhancer regions, and inhibitor regions. Each coding region indicates end of a gene’s area. All the enhancer and inhibitor regions starting at the end of the previous gene (or, in the case of the first gene, the start of the genome) up to the digit immediately preceding the indicator sequence of the coding region of the gene are considered the regulatory regions of that gene (See Figure 3.9).

![Figure 3.9](image-url)

**Figure 3.9:** An example genome. The start sequence for coding region is 010, the start sequence for enhancer is 12 and for inhibitor is 23. En and In are labels for enhancers and inhibitors, respectively, for Gene n. So the expression of Gene 1, for example, is enhanced by protein 20 (110 in base-4 notation), and inhibited by protein 8 (020 in base-4 notation).

Each enhancer or inhibitor site related to a particular gene encodes the identification number of a protein. They specify which proteins participate in regulating activation of the gene that means the connections between the nodes of the network. The coding region of a gene encodes the identification number of the protein it may produce, an output function and its related parameter, and a specification of how the produced protein is to be distributed across the different diffusion sites (See Figure 3.10).
Figure 3.10: The structure of a gene. In this example, the gene produces protein 48 (300 in base 4) when expressed. It has a Gradient Above Zero output function (2) with gradient 3 (03). The product is deposited at a specific diffusion site (3), which is site number 2 (02).

There are four different output functions that specify the concentration level of the produced protein (Figure 3.10). In each time step, the total concentration of all the proteins which are mentioned in enhancer sites of the gene minus the total concentration of all the proteins which are mentioned in inhibitor sites of the gene are calculated. The value is used as the input to the output function of the gene.

This model is implemented to control a set of robots. Based on the properties of the robot, different diffusion sites are defined for a GRN unit. When a protein is produced, it is placed in one or several diffusion sites which is specified by the data in the related gene. Protein are subject to be gradually degraded which means the concentration value of the protein decreases over time. They are also subject to be diffused between the diffusion sites within a single robot or to the other robots.

Some proteins are produced by sensors in the related diffusion sites. On the other hand, some proteins specify the behavior of the actuators of the robots. A summery of the designed controller is shown in Figure 3.11.
Figure 3.11: Summary of the GRN Controller design. For clarity, only four of the eight diffusion sites are shown.
Chapter 4

Fractal Gene Regulatory Networks
4.1 Introduction

In a series of works reported by Bentley [13, 10], Fractal Gene Regulatory Network (FGRN) is introduced. In this model, fractal protein is proposed as an abstraction of the protein substance of a gene regulatory network in an evolutionary system [14]. Fractal proteins are used as a means of interactions between genes which drives the behaviors of the FGRN system.

Robustness and efficiency of FGRN systems are investigated in [9]. They also have been evolved to perform various tasks such as producing desired patterns [10], function approximation [15], controlling conventional robots [11, 12], motion planning in non-Markovian test-bed [75], approximating the value of $\pi$ [48], pole-balancing [49] and distributed control of modular robots [74, 73].

In the following sections, first, the abstract concepts that FGRN model borrowed from the biological systems are briefly reviewed and then the components and processes in the FGRN system are explained.

4.1.1 Abstract concepts from biology

As it is described in chapter II, in the biological world, development of phenotypes can be thought of as a product of interaction between genes and proteins in their environment. Proteins drive development and functioning of a cell and are also used for communication between a cell and its environment that might include other cells.

A cell contains a genome and a cytoplasm which are surrounded by a membrane (Figure 4.1). The membrane separates the interior of a cell from the outside environment. Receptor proteins are embedded in the membrane and control the movement of environmental proteins into the cell. The cytoplasm contains a compound of proteins inside the cell. The genome consists of a set of genes. Every gene contains a sequence that encodes a protein (coding region) and a sequence that determines the conditions for activation or suppression of that gene (promoter region) (Figure 4.1).

An active gene expresses and produces its appropriate protein as encoded in its coding region. For a gene to be activated, the similarity between the protein content of the cytoplasm and the promoter region of the gene has to reach a threshold.
The cytoplasm content is altered by proteins produced by genes inside the cell or the environmental proteins which have entered the cell passing through receptors.

During the development of a cell, the protein content of the cytoplasm might match against the promoter of some genes and get them to suppress or express proteins. Every produced protein will enter into the cytoplasm and alter its content. The new content, in turn, affects the expression of genes in the next step. In this way, every protein inside a cell either produced by the genes or from environment might influence the expression of the genes directly or indirectly. On the other hand, the functional behavior of a cell is determined by particular proteins in the cell and is controlled by the cytoplasm content. All these ongoing interaction between proteins and genes continues for the whole lifetime of a cell.

### 4.2 FGRN system

An FGRN cell is a smallest independent unit in an FGRN system. An FGRN system consists of a single cell or a set of cells which run independently but capable of interacting with each other or with their environment.

The encoding part of an FGRN cell is a genome which indirectly encodes possible dynamic interactions of the system that leads to the system’s behavior. The genome consists of a series of genes encoding protein substrates and related parameters. The behavior of an FGRN cell is controlled through ongoing interactions between genome and the fractal proteins. A cell contains a so called cytoplasm which is a unit responsible for maintaining interactions between fractal proteins and genome. All the fractal proteins inside a cell is delivered to cytoplasm and merge into its protein content. The interaction be-
tween this compound of proteins and particular genes in each step, influence the content of cytoplasm by expressing more proteins and specifies the output of the cell for that step.

<table>
<thead>
<tr>
<th>Gene Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regulatory</td>
<td>Its encoded proteins are merged into cytoplasm and participate in regulation of gene expression.</td>
</tr>
<tr>
<td>Environmental</td>
<td>Determines the proteins which might be present in the environment of the cell. Environmental genes are either Maternal or Input. Maternal environmental genes are related to the proteins which are encoded to be always produced in the cell’s environment. Input genes are expressed depending on the situations in the environment.</td>
</tr>
<tr>
<td>Cell receptor</td>
<td>Encodes receptor proteins. Receptor proteins are always produced and merge together and function as a mask that permits some portions of environmental proteins to enter the cell.</td>
</tr>
<tr>
<td>Behavioral</td>
<td>These genes don’t produce any protein. The values in the coding region of these genes can directly participate in determining the outputs of the cell.</td>
</tr>
</tbody>
</table>

### 4.3 Genes

Each gene in the genome consists of different parts as follows: a promoter region, a coding region, and a set of parameters (See Figure 4.2). Both promoter and coding regions of a gene consist of three real values. These three values define square bitmap images (Figure 4.3) and enable the gene to interact with cytoplasm. Parameter region includes an affinity threshold (AT), a concentration threshold (CT), and type of the gene. Different possible types of genes are summarized in Table 4.1. Type is represented by an integer value. Both thresholds are single real values. Affinity threshold is used to determine the probability of gene activation according to protein content of the cytoplasm. It represents the allowable difference between the bitmap image of the currently existing proteins in the cytoplasm and the bitmap image defined by promoter region of the gene. Affinity threshold may be a positive or a negative value.
The sign specifies whether the gene must be activated or inactivated if the absolute difference is less than the threshold. Concentration threshold is used to determine the degree of expression of a gene when it is activated. Expression of receptor and environmental genes are independent of cytoplasm content. Therefore, promoter region and the thresholds are ignored for these types of genes.

<table>
<thead>
<tr>
<th>Promoter Region</th>
<th>Parameters</th>
<th>Coding Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_p$</td>
<td>$Y_p$</td>
<td>$Z_p$</td>
</tr>
<tr>
<td>$A_{TH}$</td>
<td>$C_{TH}$</td>
<td>$type$</td>
</tr>
<tr>
<td>$X$</td>
<td>$Y$</td>
<td>$Z$</td>
</tr>
</tbody>
</table>

Figure 4.2: A gene consists of a promoter region, a coding region, and some parameters.

The coding region contains the three real values which encode a fractal protein as a bitmap image. In the same way as the coding region, the promoter region consists of three real values that encode a bitmap image as well. This image works as a window that will be put over the bitmap image of the fractal protein content of the cytoplasm and is used to calculate the matching degree between the promoter of the gene and the cytoplasm content (See Figure 4.6 for an example).
4.4 Fractals in an FGRN system

In order to represent fractal proteins and all the interactions of them with each other and with the genome, square bitmap images are used. These images are all of a fixed pre-specified resolution (15x15 in this work).

A fractal protein is a bitmap image of a window on the Mandelbrot fractal set (Figure 4.3). It is encoded by only three real values \((x, y, z)\). \((x, y)\) specify the location of the center of a square window on the Mandelbrot fractal set. \(z\) specifies the size of the window’s sides. Having these three values and a
fixed pre-specified number (15 in this work) as resolution of sampling of the fractal set, the bitmap image of the fractal protein is generated. This image is represented by an square matrix of the pre-specified size (15x15) (For more details see Appendix I).

In association with each square matrix of a bitmap image, another square matrix exists with the same size that specifies the concentration level. The concentration level represents the current amount of a protein. All the matrix entries of the concentration matrix have the same value for each single matrix.

A desirable property of fractal bitmap images that helps evolvability of the system is the feature of self-similarity in fractal sets. Self-similarity makes redundancy in the search space. It means that the same solution (if exists and is achievable by the representation) can be encoded by indefinite number of different genotypes. The other desirable property of this representation is the simplicity of encoding. Using only three real values, the complex shape of a fractal bitmap image can be encoded. Moreover, making small changes in each of the three values leads to small changes in the fractal bitmap image and the mapping from the encoding to the encoded space is continuous.

Figure 4.4: Two bitmap square (a) and (b) are merged and make (c).

Figure 4.5: Bitmap square (a) is masked by bitmap square (b) and makes (c).
4.5 Interactions of fractal proteins

For the regulatory proteins which are produced through interaction of cytoplasm content and regulatory genes, the concentration value increases when more of the protein is produced and decreases slowly over time to resemble normal degradation that happens in biological cells.

Receptor proteins always exist with maximum concentration level and work as a mask which allows parts of environmental proteins to enter the cell.

Concentration level of the environmental proteins is determined by the environment. Environmental proteins can be either maternal which always exist or input-related proteins which change through environmental changes; for example they can be related to sensory information.

Every bitmap image related to a protein produced inside a cell or parts of a protein which enters a cell through receptors is delivered to cytoplasm to be merged with its content. Cytoplasm contains a bitmap square with the same size as the single proteins.

The interactions between bitmap images is summarized in the following paragraphs.

4.5.1 Merging

In order to calculate the merged product of two bitmap images a pixel-wise max operation is performed between the two images. It means that the winner of the operation between every two corresponding pixels is the one which is whiter (Figure 4.4). The merging operation can be shown as follows:

\[ \text{image}_R_{ij} = \max(\text{image}_{1ij}, \text{image}_{2ij}) \]  

(4.1)

Where \( \text{image}_{1ij} \) and \( \text{image}_{2ij} \) are the two bitmap images which are supposed to merge and \( \text{image}_R_{ij} \) is the resultant merged product. \( i \) and \( j \) are the co-
ordination of the pixel. The concentration value associated with the resultant pixel equals to the concentration value of the winner.

4.5.2 Masking

Each environmental protein should pass through receptors which function as a mask. Only some parts of the environmental proteins are permitted to enter the cell and be merged into the cytoplasm content (Figure 4.5). The masking operation is considered as follows:

\[
image_{R_{ij}} = \begin{cases} 
image_{1_{ij}} & \text{if } image_{F_{ij}} \neq 0 \text{ (BLACK pixel)}, \\
0 & \text{otherwise.}
\end{cases}
\] 

(4.2)

Where \(image_{1_{ij}}\) is the bitmap image of the environmental protein, \(image_{F_{ij}}\) is the bitmap image of the receptor, and \(image_{R_{ij}}\) is the product of the masking operation. \(i\) and \(j\) are the coordination of the pixel. The concentration value associated with the resultant pixel is zero if the resultant pixel is BLACK, and otherwise equals to the concentration of the \(image_{1_{ij}}\).

4.5.3 Matching

Cytoplasm content influences regulatory and behavioral genes. For regulatory genes, activation or inactivation and also the degree of protein production of the activated genes is specified through interaction of promoter region of the gene with cytoplasm content. In the same way as regulatory genes, promoter region of behavioral genes interact with cytoplasm content and specify the output value of the gene.

This interaction includes the calculation of the matching degree between the bitmap images of the cytoplasm content and the promoter of the gene (Figure 4.6) which can be formulated as follows:

\[
image_{R_{ij}} = \begin{cases} 
|image_{1_{ij}} - image_{P_{ij}}| & \text{if } image_{P_{ij}} \neq 0, \\
0 & \text{otherwise.}
\end{cases}
\] 

(4.3)

Where \(image_{1_{ij}}\) is the bitmap image of the cytoplasm content, \(image_{P_{ij}}\) is the bitmap image of the promoter of the gene, and \(image_{R_{ij}}\) is the resultant bitmap. \(i\) and \(j\) are the coordination of the pixel. The concentration value associated to the resultant pixel is zero if the resultant pixel is BLACK, and otherwise equals to the concentration of the \(image_{1_{ij}}\). All the values of the pixels in the resultant product are summed in order to calculate the matching
degree. The average of the concentration values of the resultant product is used to calculate the degree of protein expression for the gene in the case of activation.

Figure 4.7: Interaction in an FGRN cell, in every developmental cycle.

4.6 Development of a cell

Lifetime of a cell consists of consecutive developmental cycles. First of all, receptor genes produce receptor proteins. These proteins merge and make a mask for environmental proteins. Then, maternal environmental genes produce their proteins. The concentration of maternal and receptor proteins are set to maximum value by default and never change. Maternal proteins are masked by the cell receptor and the reminder enters the cell and merges with the cytoplasm content. Figure 4.7 represents the interactions of an FGRN cell in every developmental cycle. The following steps are repeated in each cycle:
1. Input signals are examined and activate the related input genes in order to produce proteins with a concentration level depending on the input signal. The produced proteins are masked by the cell receptor and some portions of them enter the cell and merge with the content of cytoplasm.

2. For each regulatory gene in the genome, matching probability of the cytoplasm content and promoter of the gene is calculated by the following formula:

\[ P_a = \frac{1 + \tanh\left( \frac{(M_D - |A_{TH}| - C_i)}{C_s} \right)}{2} \]  

where \( M_D \) is the matching degree between promoter and cytoplasm content. \( A_{TH} \) is the affinity threshold of the gene. \( C_i \) and \( C_s \) are constants and both having values of 50 in the implementation. These values are chosen such that provide a small, but distinct transitional region between probabilities.

3. For each behavioral gene in the genome, matching probability of the cytoplasm content and promoter of the gene is calculated in order to determine if the gene is activated or inactivated. Any activated behavioral gene produces a value which is calculated by the following formula:

\[ value = \pm (A_c - C_{TH}) \times x \]  

where \( A_c \) is the average concentration value calculated from the matching operation, \( C_{TH} \) is the concentration threshold. \( x \) is a part of coding region of the gene. \((+)\) takes place if the affinity threshold has a positive value and \((-)\) takes place if the affinity threshold has a negative value. The value is calculated for all of the behavioral genes and the results are summed up and make the output of the cell. In case of several outputs, another parameter can be added to the gene structure which specifies which behavioral genes are related to which cell output.

4. The concentration values for cytoplasm content are updated. Each activated regulatory gene produces its protein based on its concentration threshold and the results of matching between its promoter and cytoplasm content. The amount of the new produced protein is added to the previous concentration value. Also a degradation factor forces concentration level to decay over time. This process can be represented as follows:

\[ NewConc = P_c + (A_c \times \frac{\tanh\left( \frac{A_c - C_{TH}}{C_i} \right)}{C_p}) - (P_c + 0.2) \]  

52
where $P_c$ is previous concentration level, $A_c$ is average concentration level of matching operation, $C_{TH}$ is the concentration threshold, $C_w$, $C_i$ and $C_p$ are constants with a value of 30, 2, 5 respectively. The value of $C_p$ determines the rate of protein decay. Increasing its value will reduce protein decay, extending the time proteins remain in the cytoplasm which might make longer timings possible, but could reduce control of short-term concentration changes. Reducing the value will make proteins decay quickly, providing better short-term control, but making long-term effects harder to achieve. For the values of $C_w$ and $C_i$, the lower the values, the lower the rate of increase of protein production rate as the average concentration value increases. A concentration value more than an arbitrary maximum value (200 in this work) and less than zero will be set to the maximum value and zero respectively.

4.7 Evolving an FGRN system

Evolution searches for the fittest FGRN genomes which can drive FGRN cells to make proper behaviors for the system. In order to do that, a population of FGRN genomes is initialized randomly. The values and the number of the genes in a genome are subject of evolution. The system contains a population of genomes. Genetic algorithm is used for evolution and n fittest individuals are randomly selected as parents to produce children by crossover and mutation. The produced children replace the least fit members in the next generation. Fitness is evaluated and a time span is employed wherein individuals older than a defined time span are more susceptible to the replacement.

Crossover is performed between two randomly selected parents. That is, for each gene in one parent’s genome, most similar gene in the second parent which is still not being used in crossover is found and uniform crossover is applied. If such a gene is not found, the first parent remains unchanged in the child genome. After crossover, each gene undergoes mutation by a small probability.

Mutation can change any part of a gene including the type. Also an additional kind of mutation is performed such that with a small probability two genes inside a genome interchange their promoters or coding regions. It may happen between two promoters, two coding regions, or one gene’s promoter and one gene’s coding region. Finally, another kind of mutation can duplicate or delete a gene from the child’s genome.
4.8 Various aspects of GRNs in FGRN model

Instead of a model for studying the network properties of a GRN system, FGRN is introduced with an application-based viewpoint. As it is described in details, genome in this model is a set of genes. Along with a set of numerical parameters, every gene encodes two windows which are decoded to bitmap images from a fractal set. The network structure of an FGRN is derived based on the encoded bitmap images and the parameters of the genes. Thus, the connections between the nodes is specified through a complicated process on the genome. The model uses concentration level values. Therefore the transfer function of the nodes are not boolean functions.

As it is discussed in Chapter 3, another aspect of GRNs is multi-cellularity. FGRN is mainly designed as a single-cell model. Like other single-cell models, FGRN might be also extended to multi-cellular systems. In this thesis the system is used in a multi-cellular way (Chapter 7) in a simple form. The model may also been modifies for more complicated multi-cellular applications although it might not be done in a very straight forward way and the evolvability of the system should be considered carefully.
Chapter 5

Case studies: Sensor-coupled Fractal Gene Regulatory Networks for a Path Planning Problem
5.1 Introduction

Tartarus is a benchmark problem which is used to evaluate artificial intelligence techniques for solving problems in the field of non-Markovian agent motion planning. In this chapter a fractal gene regulatory network with inputs is evolved to act as a virtual robot controller in the Tartarus environment. The proposed technique is compared and contrasted with other previously reported techniques and it is shown that the gene regulatory network that includes input information provides an excellent performance without using any explicit memory or environmental modeling.

The proposed method may be considered as an extension to Bentley’s FGRN in that it employs input genes in a more appropriate way. Changes in sensory information cause the related input genes to be activated and express proteins while development continues normally. This continuity allows sensory signals generated by environment to influence development of the cell and contributes to the normal internal interactions. This happens without any interrupt or loss of information of life history which implicitly is stored and is represented by cytoplasm protein content.

Experiments are performed to assess the evolved FGRNs for solving the Tartarus problem [68] as a non-Markovian agent motion planning test-bed. In one experiment, FGRN without input genes is implemented. The other experiment is based on the proposed technique and the input genes are employed and results show a considerably enhanced performance compared with other techniques.

5.2 Problem Description

Tartarus is an extension, with a more difficult task and more challenging spatial modeling, of artificial ant problem [42]. Tartarus environment is a grid board surrounded by walls. There are six square blocks, each occupy one grid square and are placed randomly in the environment away from the walls. The board is never initialized such that four blocks form another square (as shown in Figure 5.1a which is an invalid board configuration whereas Figure 5.1b is a valid board configuration). A virtual agent is also placed away from the walls in the environment with position and facing direction initialized randomly.

The agent senses only 8 adjacent grid cells. Agent’s sensors detect whether a cell is empty; occupied by a block; or by a part of a wall as shown in Figure
5.1c. In each step it may turn left, right, or move ahead. The agent may push only one block ahead in each move. There is no limit on time, it has only 80 units of energy and is allowed a total of 80 moves and/or rotations. It means that the agent may stay motionless and think without losing energy level. The agent’s goal is to place as many blocks as possible to the sides against the walls. A block in a corner scores two while a block against a wall away from the corner scores one. Thus the maximum possible score is ten for four blocks in the corners plus two blocks against the walls. An example is shown in Figure 5.2.

5.3 Computational Procedure

5.3.1 Previous methods

Different methods have been applied to the Tartarus problem by several authors. In [68], indexed memory is added to GP technique to incorporate the evolution of gathering, storage, and retrieval of arbitrarily complicated
state information and creating a mental model. In the general case, each individual in the population has a functional tree, an integer array of elements (memory) indexed from zero to (M-1). In [6], neural networks are applied to the Tartarus problem. The authors argue that designing the sensory system for a robot combined with appropriate controllers can lead to improvements in performance. [5] employs co-evolution for an implementation of Genetic Programming automation (GP-automaton). The GP-automaton is essentially a finite state machine augmented by a parse tree representing an expression for each state. The formula is used to interpret inputs for the automaton. Each GP-automaton consists of a collection of states and their associated expressions, including a distinguished initial state, a transition and a response function. The transition function is used to determine the next state of the GP-automaton and the response function computes the output of the GP-automaton. [25] reports the same representation and modified operators using four different co-evolutionary algorithms. In [4], a GP-based method uses GP-automata to design controllers operating on three front sensors for an agent. Another GP-based algorithm in [3] implements If-Statement-Action (ISAc) list. ISAc table is a list of very simple instructions, called nodes, executed orderly except when a jump modifies the order of execution. In [2], authors developed a finite state machine, serving as a controller that translates sensors information into actions, using Genetic Algorithm (GA). They managed to use a pure Finite State Automata (FSA) gene for the Tartarus problem by augmenting the action set of the FSA controller. These actions specify which move the controller should make, and specify which of the eight available inputs will drive the next finite state transition. This permits the FSA to evolve a strategy for using the available information. In [46] a tree state machine is implemented, which has an evolving tree structure for sensor-motor mapping and encodes internal states. The work demonstrates that both state sensors and memory states are important and influential factors in performance of the agent. In [70] a long-term memory is utilized. The long-term memory is an 11x11 grid which is large enough to allow data sensed from the agent’s initial position to be placed in the center and then a map will be produced from this point. Also GA is used within the agent to evolve command sequences that may be carried out. A fitness function evaluates a chromosome by simulating the execution of the command sequence using a copy of the map contained in the memory. After completing the route, the final score (based on Tartarus scoring policy) is added to the fitness and weighted by a factor of 100.
Table 5.1: Comparison of the previous methods applied to the Tartarus problem.

<table>
<thead>
<tr>
<th>Method</th>
<th>Score achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4 Indexed memory [68]</td>
<td></td>
</tr>
<tr>
<td>4.5 Neural network [6]</td>
<td></td>
</tr>
<tr>
<td>7.8 Co-evolution of GP-automaton [5]</td>
<td></td>
</tr>
<tr>
<td>7.2 Other Co-evolutionary algorithms [25]</td>
<td></td>
</tr>
<tr>
<td>8.15 GP-automata with three front sensors [4]</td>
<td></td>
</tr>
<tr>
<td>8.2 ISAc list [3]</td>
<td></td>
</tr>
<tr>
<td>7.11 FSA [2]</td>
<td></td>
</tr>
<tr>
<td>7.99 Tree state machine [46]</td>
<td></td>
</tr>
<tr>
<td>4.38 Long-term memory - simple [70]</td>
<td></td>
</tr>
<tr>
<td>8.77 Long-term memory with heuristics [70]</td>
<td></td>
</tr>
<tr>
<td>7.6 FGRN without input (this work)</td>
<td></td>
</tr>
<tr>
<td>8.2 FGRN with input (this work)</td>
<td></td>
</tr>
</tbody>
</table>

to create a final fitness for the command sequence. The method is applied to a simple case as well as applying exhaustive search and advanced heuristics. Applied heuristics are problem-dependent with little generalization capability. A comparison of the above mentioned methods with the proposed techniques (in terms of scores achieved) is shown in Table 5.1 in the experimental section.

5.3.2 Proposed method

In our work, input signals influence interactions of internal proteins in a FGRN. To drive development of a cell, the proposed method blends sensory and genetic information. Sensory information is received from environment of the cell and genetic information is presented by the cell’s genome. Environmental signals trigger input genes causing the expression of corresponding proteins. Therefore, in each developmental state, the cytoplasm content and consequently devolvement trace of the cell depends on both the cell’s genetic information and environmental signals. In addition to employment of input genes in FGRN in a more appropriate way, genes are augmented by an additional real value called subtypeID that relates input genes to specific sensory information. In the case of input genes, crossover occurs between genes with the same identifier. SubtypeID is also used for mutation for relating the gene to different sensory signals. Differentiation between triggered sources of input genes helps evolution by reducing the search space. It allows evolution to select more
Behavioral genes are separated into two different sets based on their subtypeID; calculated values of genes inside each set are summed and produce two outputs (Output$_a$, Output$_d$). Outputs determine robot action.

<table>
<thead>
<tr>
<th>Output$_a$</th>
<th>Output$_d$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>≠</td>
<td>Move ahead</td>
</tr>
<tr>
<td>0</td>
<td>#</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Turn right</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Turn left</td>
</tr>
</tbody>
</table>

**Figure 5.3:** Behavioral genes are separated into two different sets based on their subtypeID; calculated values of genes inside each set are summed and produce two outputs (Output$_a$, Output$_d$). Outputs determine robot action.

useful sensory information and consequently a more efficient solution to the problem.

Initial population is formed randomly and each individual is a Tartarus robot which contains an empty cytoplasm FGRN cell with a randomly generated genome. The cell is a decision center for robot. It receives signals from environment of the robot and generates outputs in each step. Robot acts on the basis of these outputs.

The cell contains a genome which keeps genetic information and a cytoplasm which keeps proteins for creating an internal state of the cell. The genome consists of genes and each gene has a type that determines its role in the cell. Depending on its type, a gene may be activated by input signals received from the environment, current protein levels in cytoplasm, or always be activated and produce corresponding protein.

Tartarus robots have 8 sensors each senses one adjacent grid in the environment. Each sensor has one of the three values: the grid is empty; the grid is occupied by a block; or it is a part of a wall. An individual receives the sensors information and makes input signals for its cell. Each sensor can issue one of the two different related input signals or it might issue no signal. If the sensor identifies a part of wall or a block, it issues a signal that represents this
information and if it identifies an empty grid no signal is issued (Figure 5.4). In total 16 different input signals exist, each of which triggers corresponding input genes in the cell genome if there is any. Corresponding input genes are identified by the subtypeID of the genes. A subtypeID of a value between zero to 15 determines the input signal which can trigger the gene. Therefore, several input genes may be triggered by one input signal. On the contrary, some input signals do not trigger any input genes. SubtypeID allows evolution to regulate the number of input genes for each input signal and the less important signals are not associated with any genes.

In addition to sensing the environment, Tartarus robots should take an action in each step. Robot actions are calculated based on cell outputs. Behavioral genes of a cell are separated into two different sets based on their subtypeID and each set is related to one cell output. Every activated behavioral gene represents a value and sum of these values in each set composes the corresponding output for the cell (Figure 5.3).

Structure of a gene is shown in Figure 5.5. Each part of a gene in a cell is initialized by a random number. Values of promoter and protein regions are initialized by a random real number between -1 and 1. Affinity threshold is initialized by a random value between -10000 and 10000. Concentration threshold is initialized by a random value between zero and 200. Type is initialized by a random value between zero and four and subtypeID is initialized by a random value between zero and 15 for input genes, zero or one for output genes and zero for other genes.

In each generation, new individuals undergo 80 developmental cycles to generate 80 movements required by the Tartarus problem. At the end of 80 cycles the individual’s fitness is evaluated based on its performance in the environment.

Development of each individual consists of several steps. In the first step, receptor genes produce corresponding proteins. These proteins merge and construct a mask for environmental proteins called cell receptor and only permit some parts to enter the cell cytoplasm. Then, maternal genes produce their proteins and concentration of these proteins is always 200 (maximum value) which means that these proteins are saturated. Maternal proteins are masked by the cell receptor and the reminder merges and enters the cytoplasm. The following tasks are repeated in each developmental cycle:

- Input signals are examined and the related input genes for each received input produce their proteins with concentration level of 200. They are
Figure 5.4: Sensory information issue input signals which activate corresponding input genes.
Figure 5.5: Structure of a gene.

masked by the cell receptor and the reminders enter the cytoplasm and merge with its content.

- For each regulatory gene in the genome, the cytoplasm content is compared with the fractal subset (square shaped) that the gene promoter region represents. The difference determines whether the gene must be activated or not. Matching probability is calculated by the following formula:

\[
Pa = \frac{(1 + \tanh((\text{difference} - |affinityThreshold| - C_t)/C_s))/2}{2} (5.1)
\]

where difference is the result of comparison between promoter and cytoplasm content \( C_t \) and \( C_s \) are constants and both having values of 50 in the implementation.

- For each behavioral gene in the genome, the promoter is compared with the cytoplasm content of the cell. The gene is activated based on the matching degree. Any activated gene produce a value which is calculated by the following formula:

\[
\text{value} = \pm (\text{totalConcentration} - \text{concentrationThreshold}) \times x \quad (5.2)
\]

where \( \text{totalConcentration} \) is the average concentration level of all sampling points of cytoplasm content which are matched with the gene promoter (are located in non-black area). \( x \) is a real value in protein region of the gene. (+) takes place if the affinity threshold has a positive value and (-) takes place if the affinity threshold has a negative value.

The \( \text{value} \) calculated for the behavioral genes with zero subtypeID are summed. The summation forms one of the cell output. The same procedure is performed for the behavioral genes with subtypeID1 and the second output of the cell is created. Therefore, each cell produces two outputs and an individual calculates its action on the basis of these outputs.
• Each activated regulatory gene produces its protein based on its concentration threshold and the cytoplasm content. The produced protein is merged with cytoplasm content and might influence activation of genes in the next cycle. A new value is calculated for the concentration level of the protein contained in the cytoplasm. The amount of the new production is added to the previous concentration value. Also a degradation factor forces protein level to decay over time. This process can be represented as follows:

\[
newConc = Pc + (Tc \times \tanh((Tc - ct)/Cw)/Ci) - (Pc/Cp + 0.2) \quad (5.3)
\]

where \(Pc\) is previous concentration level, \(Tc\) is total concentration level, \(ct\) is the concentration threshold, \(Cw\), \(Ci\) and \(Cp\) are constants with a value of 30, 2, 5 respectively.

• Robot takes an action on the basis of calculated outputs. If the first output has a positive value, the robot moves. Otherwise, the second output is considered and the robot turns right if it has a positive value, and turns left otherwise. After completion of all 80 developmental cycles, fitness is evaluated for each individual. The population is sorted in descending order of individual’s fitness. In addition to the fitness value, each individual in the population has an age value that represents the number of generations it has participated in the population. If the age of an individual is more than a specified threshold (ten generations), then that individual losses its place and moves to the end of the population where it may be removed in the next generation. This lifespan threshold allows more exploration.

To create a new generation, top 40 individuals in the population are selected and placed in the mating pool. Two randomly selected parents generate two children after crossover and mutation. Children are placed in the Tartarus environment and pass developmental cycles and their fitness is evaluated. Then each child searches the population from top to find an old individual with less fitness. If such individual is found, new individual is inserted and one individual in bottom of the list is removed. Otherwise, the new individual is ignored and population stays unchanged. The flowchart in Figure 5.6 summarizes the complete procedure.
Figure 5.6: Complete procedure.
5.4 Simulation Results and Discussion

Two different experiments are carried out to investigate FGRN performance in solving the Tartarus problem. The progress and behavior of the proposed algorithm as well as the final solutions are analyzed in detail. The effect of the sensory information and reasons for similarities and differences between individual’s solutions in one converged population is thoroughly investigated. Table 5.2 shows all parameters used in both experiments.

In the first experiment a population of sensor-less individuals is evolved to find a solution for the Tartarus problem. Each individual genome in this experiment is initialized with randomly generated regulatory, receptor, maternal, and behavioral genes. Since individuals are sensor-less, input genes are excluded in this experiment. Initial number of each type of genes is shown in Table 5.2. In each developmental step, behavioral genes produce 2 outputs which are mapped to the three robot commands. FGRN is allowed to regulate the number of each type of genes during evolution.

The experiment is repeated for 100 runs and at the beginning of each run the board configuration is randomly initialized. A population of 100 individuals is allowed to evolve within 500 generations. In each generation the best 40 individuals participate in crossover and mutation to produce 80 children. After fitness evaluation each child may replace an old individual in the population. Crossover and mutation are always applied. A fitness function is defined to evaluate the score of final board configuration based on score policy of Tartarus problem. Also a less important fitness function with a production factor of 0.01 is employed which scores each successful block move (see Table 5.2). The Tartarus score achieved for each board configuration is averaged over 100 runs and a score of 7.6 is obtained.

In the second experiment, a population of sensor-enabled individuals, as proposed, is evolved. Input genes are included in the individual’s genome. Two input genes are associated with each agent’s sensor. Therefore, 16 input genes for each subtypeID are initialized (see Figure 5.7). One input is activated when a block is detected and the other is activated when a part of the wall is detected. When a sensor finds an empty grid cell, both related genes are inactivated. The relation between input genes and input sensor’s information is established by the subtypeID. Evolution can dynamically modify the number and the subtypeID of all genes in the genome and consequently it can adapt input genes for most useful sensory signals and improves perfor-
Table 5.2: Parameters which are common in the two experiments.

<table>
<thead>
<tr>
<th># runs</th>
<th>population size</th>
<th># generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
<td>500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># development cycles</th>
<th>crossover rate</th>
<th>mutation rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>40%</td>
<td>1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># regulatory genes</th>
<th># receptor genes</th>
<th># maternal genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># behavioral genes</th>
<th># input genes</th>
<th>Fitness function</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0-16</td>
<td>Tartarus score + 0.01 * # block moves</td>
</tr>
</tbody>
</table>

Figure 5.7: Input genes with subtypeID of number 2i and number 2i+1 are related to sensor[i].

Performance of the system. All other parameters for this experiment are the same as shown in Table 5.2. This experiment obtained an excellent score of 8.2 over 100 runs for randomly generated board configurations. The convergence curves are shown for both experiments in Figure 5.8, and Table 5.1 shows the comparison of scores achieved by the methods reported by other researchers and the techniques proposed here.

5.4.1 Progress of the Algorithm

A population of 100 sensor-enabled individuals with parameters shown in Table 5.2 are evolved for a sample board configuration shown in Figure 5.9a. Evolution found one solution after about 200 generations with the maximum score of ten, and continued until all individuals in the population converged to this maximum score. It was observed that every individual solution represented the same sequence of robot commands, shown in Figure 5.9c and final
board configuration shown in Figure 5.9b. The gray squares in Figure 5.9c mean move ahead, R turn right, and L turn left.

To have a better understanding of the mechanism applied by the cell to solve the problem, different inner quantities of the solution cell such as protein concentration levels and gene activation patterns are also inspected. Figure 5.10 shows the sequence of concentration levels of the matching between cytoplasm content and promoter of the behavioral genes. The darker color of a square in this figure represents higher concentration level. The sequence of overall produced outputs of the two sets of behavioral genes and the resultant robot command sequence are shown in Figure 5.11. The black squares in the sequence of the produced outputs mean positive output and white squares negative output.

The concentration level of regulatory proteins corresponding to each regu-
latory gene is shown in Figure 5.12. Similar patterns can be recognized in the sequence of the two produced outputs (Figure 5.11) and the regulatory protein concentration levels (Figure 5.12). As Figure 5.12 shows, the cell contains four regulatory genes, but only three are participating in the developmental steps of this particular solution cell and the regulatory gene numbered zero seems to be redundant, this will be investigated later.

As previously explained, the input genes are activated or deactivated on the basis of sensory information. Figure 5.13 shows the activation pattern of input genes of the solution cell under consideration. Black squares represent activation and white squares deactivation. Figure 5.13 shows two genes with subtypeID 7 and two genes with subtypeID 15. Input genes with subtypeID of 5, 8, 10 and 12 are absent in the genome of this certain solution cell.

**Figure 5.10:** Sequence of concentration levels of the matching between cytoplasm content and promoter of each behavioral gene

**Figure 5.11:** Sequence of produced outputs and corresponding robot command sequence.

**Figure 5.12:** Concentration level of regulatory proteins corresponding to each regulatory gene.
5.4.2 Effects of input information

In an experiment, all sensors are inactivated and therefore the input genes are prohibited to make proteins and do not participate in development. These modified individuals are placed on the board with the same configuration as shown in Figure 5.9a. The evaluation of population show three groups of individual’s score: one individual with a score of four; 42 individuals with a score of three; 57 individuals with a score of zero. For the purpose of comparison, the individual with the score of 4, namely cell 4 and a randomly selected individual with a score of zero, namely cell 0 are chosen. Figures 5.14 and 5.15 show the sequence of concentration level of matching between cytoplasm content and the promoters of the behavioral genes for these two individuals. Also, the sequence of produced outputs, robot commands and the sequence of concentration levels of the regulatory proteins in developmental steps are shown in Figures 5.14 and 5.15. These figures highlight patterns emerging with noticeable differences in their genomes that may have been hidden in the presence of input proteins. For example, small differences exist in their maternal factors as shown in Figures 5.18 and 5.19. However, Figures 5.14 and 5.15 show different behavioral and regulatory patterns. These figures also show a different kind of participation in developmental process for regulatory genes between the two selected cells and also in comparison with Figure 5.12, the case of sensors enabled. It is observed in Figures 5.14 and 5.15 that cell 4 uses regulatory genes number 1 and 2 and cell 0 uses regulatory genes number 0 and 2. Recall that the regulatory gene number 0 seemed to be redundant in
development of the cells in the last observation (Figure 5.12). This gene which was apparently left out from the process of evolution is now used in this new circumstance for cell 0.

It is clear that sensor-enabled genes are valuable ingredients. Figures 5.16 and 5.17 show input proteins of the two selected cells. Since the input genes with the same subtypeID are activated in the same situation, their corresponding proteins are merged as shown in Figures 5.16 and 5.17. As explained previously, maternal factors are always present in cytoplasm and receptor proteins mask input proteins which are then merged with the maternal factors. Figures 5.18 and 5.19 represent maternal proteins, receptor proteins, and the effective parts of the input proteins. Dashed areas in these figures represent ineffective areas of the input proteins and the dark areas represent regions with a very low affect. It is observed that out of 16 genes, the genes numbered 5, 8, 10, 12 have been eliminated during the evolution, the effects of gene number 0 is almost negligible and the shape of proteins are different for all other genes.

**Figure 5.14:** a) The sequence of concentration level of matching between cytoplasm content and the promoters of the behavioral genes, b) the sequence of produced outputs and robot command, c) the sequence of concentration level of the regulatory proteins of the cell 4.
Figure 5.15: a) The sequence of concentration level of matching between cytoplasm content and the promoters of the behavioral genes, b) the sequence of produced outputs and robot command, c) and the sequence of concentration level of the regulatory proteins of the cell0.

Figure 5.16: input proteins of cell4.
**Figure 5.17:** Input proteins of cell0.

**Figure 5.18:** Receptor, maternal, and effective areas of the input proteins of cell4.
Figure 5.19: Receptor, maternal, and effective areas of the input proteins of cell0

5.5 Conclusion

In this chapter a gene regulatory network is employed to solve the Tartarus problem. Since it is a non-Markovian motion planning, a kind of memory is necessary. The fractal gene regulatory network with input which is successfully implemented here, equips the agent with an implicit memory. This makes the agent capable of regulating actions that are contributed from input signals received from the environment. Two experiments are performed. In the first experiment FGRN without input genes is evolved and reached a good result of 7.6. In the second experiment the proposed method that allows input signals to contribute to the cell interactions is implemented and an excellent score of 8.2 is achieved. The performance of the FGRN with input genes are compared and contrasted with other techniques and remarkably enhanced results are obtained.

The behavior of the proposed algorithm and different phenomena occurring during the processes of evolution and development of FGRN are experimentally investigated. One interesting phenomenon is that at one stage the genome of a cell which seemed to be redundant, later participated and became useful under different condition. Furthermore, the effect of sensory information, similarities and differences between structures of the solution cells in one converged
population is experimentally highlighted.

Since the method is problem independent, it can be used to evolve complex solutions with a minimum modification for different problems.
Chapter 6

Case studies: Investigating Evolutionary Behavior of Fractal Gene Regulatory Networks in Presence of the Best Solution from the Past Generations
6.1 Introduction

This chapter is concerned with behavior of an evolving population of GRNs capable of sensing the output sequence (pattern) which is generated by the best evolved individual. A long pattern generation problem is solved by the system. A population of Fractal Gene Regulatory Networks is evolved to solve the problem in a special case. The capability of sensing the best generated pattern is added to the model such that input proteins produced by activated sensors are applied in order to send the best already generated sequence of output values (pattern) to the individual’s cell.

In each generation every gene regulatory network in the population develops for a specific number of steps and produces an output value in each step. The sequence of these values is considered as a pattern which is generated by that individual. At the end of each generation, the fittest generated pattern is identified and exposed to the population such that individuals capable of sensing that pattern in the next generation.

Detailed experimental studies are performed to investigate the evolutionary behaviors of this system and it is observed that an imitational behavior spontaneously emerges through evolution as a consequence of possessing this capability while some level of creativity is preserved. This system is compared with an ordinary population of fractal gene regulatory networks and enhanced performance is demonstrated.

6.2 Problem Description

A population of Fractal Gene Regulatory Networks is evolved to solve a pattern generation problem in a special case. The purpose is to investigate the behavior of an evolving population of GRNs which is capable of sensing the output sequence (pattern) which is generated by the best evolved individual. A long pattern generation problem is solved by the system. The capability of sensing the best generated pattern is added to the model such that input proteins produced by activated sensors are applied in order to send the best already generated sequence of output values (pattern) to the individual’s cell. In each generation every gene regulatory network in the population develops for a specific number of steps and produces an output value in each step. The sequence of these values is considered as a pattern which is generated by that individual. At the end of each generation, the fittest generated pattern is
identified and exposed to the population such that the individuals are capable
of sensing that pattern in the next generation. Effects of the added capability
and comparing the achieved results and the result from the normal method
are subjects of investigation of this work.

6.3 Computational Procedure

A population of arbitrary 160 FGRN individuals is employed to solve the long
pattern generation problem. The desired pattern is a sequence of 100 two-digit
binary values. The sequence consists of 25 binary values of 00, 25 binary values
of 01, 25 binary values of 10, and 25 binary values of 11 orderly. FGRNs are
expected to generate the desired pattern.

In each step of development, the corresponding value of the already fittest
generated pattern is sensed. If any input gene corresponds to the sensed value it
will be triggered and express its protein. The protein enters the cytoplasm after
being masked by receptor proteins and then may influence the development in
addition to other proteins in the cytoplasm.

At the end of each developmental cycle, the output value of each FGRN
individual is calculated. This value is determined by its behavioral genes. Be-
havioral genes are separated into two different sets based on the subtype value
of the gene. Each set is related to one output digit. As mentioned before, all
genes in a genome contain a protein region which is represented by three real
values. Only the first value of the three (x value) is used in all the activated
behavioral genes in order to compute the gene’s offered value for this problem.
Sum of the offered values in each set generates output for the corresponding
developmental cycle. If the sum is greater than zero, the corresponding digit is
1, otherwise the digit is 0. Once 100 consecutive developmental cycles are per-
formed, the generated output pattern of the individual is ready to be compared
with the desired pattern and the corresponding fitness value be calculated.

Initial population is formed randomly and there is no fittest generated
pattern by any individual. Each part of a gene in a cell is initialized by a
random number. Values of promoter and protein regions are initialized by a
random real number between -1 and 1. Affinity threshold is initialized by a
random value between -10000 and 10000. Concentration threshold is initialized
by a random value between zero and 200. Type is initialized by a random
integer between zero and four and subtype is initialized by another random
integer between zero and four for input genes, zero or one for output genes and
zero for others.

Population evolves for 200 generations. In each generation, new individuals undergo 100 developmental steps to generate an output pattern required by the problem. When development is completed the individual’s fitness is evaluated based on the fitness function. The fitness function simply counts the number of correct digits chosen by the individual through all developmental steps. Since each individual in each developmental step chooses a two-digit binary value, the maximum possible fitness value is 200.

To create a new generation, top 40% of the population are selected and placed in the mating pool. Two randomly selected parents generate two children after crossover and mutation. Children are placed in the environment and undergo developmental steps and their fitnesses are evaluated. Then each child searches the population from top to find an old individual with less fitness. If such individual is found, new individual is inserted and one individual in bottom of the list is removed. Otherwise, the new individual is ignored and population stays unchanged. The flowchart in Figure 6.1 summarizes the complete procedure.
Figure 6.1: Complete procedure.
Table 6.1: Parameters which are common in both experiments.

<table>
<thead>
<tr>
<th>#runs</th>
<th>population size</th>
<th>#generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>160</td>
<td>200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#development cycles</th>
<th>crossover rate</th>
<th>mutation rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>40%</td>
<td>1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#regulatory genes</th>
<th>#receptor genes</th>
<th>#maternal genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#behavioral genes</th>
<th>#input genes</th>
<th>Fitness function</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0-4</td>
<td>Number of correct digits</td>
</tr>
</tbody>
</table>

### 6.4 Simulation Results and Discussion

In order to investigate the capability of sensing the fittest generated pattern in the population, three experiments are carried out. All experiments employ the same desired pattern and similar genetic configuration, as shown in Table.6.1. The first experiment employs ordinary FGRN without any information in its environment. In the second experiment, individuals include sensors and input genes providing the capability to see the fittest already generated pattern in the current step. The third experiment differs from second experiment in that regulatory genes are eliminated for comparison purposes. Statistical tests are performed to justify the results and comparisons.

![Fitness curves](image)

**Figure 6.2:** Fitness curves of the simple FGRN and the FGRN sensing the fittest pattern.
6.4.1 First Experiment

In the first experiment, all FGRNs initially consist of 2 behavioral genes, 2 receptor genes, 4 regulatory genes, and 1 maternal gene (Table.6.1). Since there is no sensory information, no input gene is employed. The population is evolved for 200 generations and the experiment is repeated for 100 runs and a mean fitness value of 173.28 is obtained as shown in Figure 6.2.

![Fitness curve of the FGRN sensing the fittest pattern (fitness/10), effectiveness curve of input genes on behavioral genes, and effectiveness curve of regulatory genes on behavioral genes.](image)

6.4.2 Second Experiment

In the second experiment, all FGRNs initially consist of 2 behavioral genes, 2 receptor genes, 4 regulatory genes, 1 maternal gene, and 4 input genes (Table.6.1). An individual has four sensors which senses the fittest already generated pattern in the appropriate step and get the appropriate input genes to be expressed. The fittest already generated pattern is the pattern with the maximum achieved fitness which is generated by an individual during all the past generations. Therefore, at the end of each generation, if any promotion in the fitness has occurred, the fittest of already generated pattern has been updated. Anytime when a new fitter pattern is generated, old individuals must be re-evaluated in the next generation just like a newborn. This is due to the fact that the new sensory information is ready and may change their fitness.

In each developmental step, the corresponding value of the fittest already generated pattern is exposed to all members of population. This can be one of the four possible values: 00, 01, 10, 11. Each value triggers one of the individual sensors and each sensor activates one group of input genes corresponding to
that sensor. Initially, only one input gene corresponds to each sensor, but it can be arbitrarily decreased or increased during the evolution. Population evolves for 200 generations. The experiment is repeated for 100 runs and a mean fitness value of 189.1 is achieved which is a significant improvement over the first experiment (see Figure 6.2). Student t-test is performed and shows that the improvement is statistically significant with 99% confidence.

Figure 6.4: Imitation error and fitness curve.

6.4.3 Third Experiment

The third experiment is similar to the second but here the regulatory genes are excluded from the model to investigate whether the regulatory genes really participate in the system with the capability of seeing fittest already generated pattern. The experiment is repeated for 100 runs and a mean fitness value of 125 is reached which is considerably lower than the prior experiments meaning that the regulatory genes really participate in the system.

6.4.4 Which proteins are effective on a gene in a specific developmental cycle?

FGRN is complex by its nature. In fact, activation or deactivation of a gene in a FGRN depends on the combination of shape and amount of many existing proteins in cytoplasm. While input signals are permitted, any change in an input signal might change all of a developmental path. As a result, derivation of the network structure of a FGRN is a difficult problem. One hard thing for creating a network structure - even in one specific developmental step - is
that in addition to the effects of the number of pixels of cytoplasm which are matched by a gene’s promoter, concentration level of a single pixel can also have a significant effect on the amount of activation of that gene. Therefore, even if at one developmental step no new protein is activated in cytoplasm, any decrease or increase in concentration level of the currently existing proteins can change the fate of a protein in the next step.

As explained before, cytoplasm content is a combination of expressed proteins of regulatory, input, or maternal genes with their specific concentration levels. On the other hand, any regulatory or behavioral gene can be activated and might express its appropriate protein on the basis of the content of cytoplasm in the regions which overlap the gene’s promoter. Both the number of pixels in the overlapped region which match the gene’s promoter and the concentration level of those pixels can effect the expression of that gene. While each pixel in cytoplasm content belongs to one protein, activation or deactivation of a gene might be dependent on several proteins which are overlapped with their promoters.

Based on this analysis we used a simple rule of potential effectiveness to determine whether a regulatory or input protein effects (and to what extent) the behavioral gene. The applied rule is as follows: All existing proteins which are not hidden by others in cytoplasm and have overlapped the behavioral gene’s promoter in any developmental step are simply considered as effective proteins for that gene. Although it might not be a very complete rule for determining the dominant effective proteins for a gene in a specific developmental path, it trims the proteins which have certainly no effect on the gene and determines the potentially effective proteins. Also, this rule is simple enough to be implemented.

6.4.5 Imitational behavior emerges: While the fittest found pattern is visible, what technique the GRN individuals employ to solve the problem?

To have a general picture of how strong the behavioral genes in the population solutions are dependent on input and regulatory genes, we applied the simple rule of potential effectiveness at the end of each generation and averaged it over all members of population. The average of total input gene’s effects on all the behavioral genes, the average of total regulatory gene’s effects on all the behavioral genes, and the averaged fitness of the best individuals in the
population during 200 generations are shown in Figure 6.2. The statistical significance of the increment and decrement of the gene’s effects which is reached by the evolution (as shown in Figure 6.3) is supported by performing a t-test with %99 confidence.

Note that, the initial number of both input and regulatory genes in a genome is the same and is set to four and evolution is free to modify this number. As Figure 6.3 illustrates, the behavioral genes depend strongly on input genes in an increasing manner up to about 8 units. On the other hand, the effect of regulatory genes on the behavioral genes is almost high in early generations but decreases gradually to the small value of about 2 units.

Based on these evidences it can be claimed that when enough generations have passed, individuals start to imitate what they have observed from the results generated by their superior colleague albeit with few modifications.

In order to elaborate this claim further, the behavior of the evolving population is observed during generations. With this in mind, 100 random patterns of length 100 (equal to the number of development cycles) are generated. A population is evolved for 200 generations. At the end of each generation, the fittest individual is chosen. The individual is re-evaluated with new conditions. For each of the 100 random generated patterns, the individual is restarted and
Figure 6.6: Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, desired pattern represented to the individual, and the difference between the generated pattern and the environmental pattern for the first selected individual.

fully developed for 100 steps while the random pattern is presented to the individual as the fittest already generated pattern. Once development completed, the pattern generated by the individual is contrasted with the presented random pattern and the number of differences between the two patterns is counted as imitation error. The average result of this evaluation for all the random patterns is calculated for every best individual and considered as the total imitation error in every generation. The experiment is repeated for 100 runs and the total imitation errors are averaged over all runs. As Figure 6.4 shows, the average imitation error is about 75% in the first generation and is decreased in the proceeding generations and reaches about 21% in the last generation. This significant decrement is supported by a confidence of 99%. 18 of 100 runs reached the exact value of zero, which are the cases that the individuals were merely imitators. These results reveal that, evolution tends to generate individuals who are almost good imitators, however, in most cases the ability of innovation is not completely vanished. To give a better picture of inner dynamics of an evolved solution, in another experiment, two typical fittest individuals with different imitation errors are selected from two different evolved populations. The fitness and imitation error is respectively 192 and 3% for the first individual and 196 and 30% for the second individual.
individuals are developed separately while different patterns are presented to the individuals of the best generated pattern. Sequences of concentration levels of input, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, the pattern that is presented as the best generated pattern and the presented pattern are illustrated in Fig 6.5 to Figure 6.10. Figure 6.5 represents the pattern generated by the first individual against the real fittest already generated pattern which is evolved by the population. Figure 6.6 represents the pattern generated by the first individual against the pattern which is desired by the problem. Figure 6.7 represents the pattern generated by the first individual against a randomly generated pattern. Figure 6.8, Figure 6.9, and Figure 6.10 represent the patterns generated by the second individual against the same patterns mentioned for the first individual. As it is shown in the figures, in the first evolved individual, regulatory genes have no effect in behavior of the cell. This effect is almost low for the second individual. On the other hand, behavioral genes are mostly dependent on the input genes in both examples. This situation leads to imitational behaviors of both individuals.
Figure 6.8: Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, fittest generated pattern presented to the individual, and the difference between the generated pattern and the environmental pattern for the second selected individual.

Figure 6.9: Sequences of concentration levels of input proteins, regulatory proteins, activation levels of behavioral genes, decision sequence of the individual, desired pattern presented to the individual, and the difference between the generated pattern and the environmental pattern for the second selected individual.
6.5 Conclusion

In this chapter, a population of FGRNs augmented with input genes and capable of seeing the fittest already generated pattern is considered for solving a long pattern generation problem. The effect of adding the capability to the system in order to improve the performance is demonstrated comparing ordinary FGRN system. Additional experiments are performed and indicate that the effects of regulatory genes on the behavior of the fittest individuals are higher in early generations and decreases over time. On the other hand the influence of input genes increases over time and when enough generations are passed the behavior of individual become more dependent on input genes than the regulatory genes. This might lead to an imitational behavior. The imitational behavior from any pattern which is presented to individuals is investigated by experiments. It is recognized that evolution encourages this behavior over generations while better solutions are being found for the problem. Individuals in the evolving population become increasingly good imitators while they can gently modify the best patterns of their society.
Chapter 7

Case studies: Fractal Gene Regulatory Networks for Robust Control of Modular Reconfigurable Robots
7.1 Control of Modular Robots

7.1.1 Introduction

Modular self-reconfigurable robots are distributed robots made up from a number of mechanically coupled modules where each module is typically controlled by its own local controller. These robots are distributed and dynamic by nature and they have limited inter-modular communication and processing capabilities. Designing controllers for modular robots is difficult due to the distributed and dynamic nature of the robots. In this chapter fractal gene regulatory networks are evolved to control modular robots in a distributed way.

![Figure 7.1: Examples of different modular robots](image)

7.1.2 Modular Self-Reconfigurable Robots

Module self-reconfigurable robots are built from modules, which are a kind of robotic cell (See Figure 7.1 for some examples) [65].

The nature of being modular is to encapsulate some of the complexity of the functionality of a module. This means that while regular screws are not modules, a drilling machine is. However, in order to be part of a modular robot,
a drilling machine has to be connected to at least another module, e.g., a robot arm. In a modular robot, each module is a simple robot containing all the on-board components required to create a robot: actuators, sensors, batteries, and processing power. In addition, a module has a way to communicate with other modules and active connectors that allow it to connect to neighboring modules and disconnect from them again. The on-board actuators allow a module to move itself with respect to connected, neighboring modules or to move a neighboring module. This, in combination with the active connectors, allows a module to wander around on a structure of modules by going through sequences of disconnect, move, and connect operations. Since all the modules of the robot can do this, the robot as a whole can change its shape. This ability to change shape is what sets self-reconfigurable robots apart from all other types of robots.

Modular self-reconfigurable robots are of different types: Lattice-type robots are a type of modular robots in which modules are organized in a lattice structure similar to the way atoms are organized in a crystal. The exact implementation of a self-reconfiguration step varies from robot to robot, owing to differences in module geometry, number of connectors, degree of freedom, etc., but fundamentally a self-reconfiguration step is based on a sequence of disconnect, move, and connect.

Another type of modular self-reconfigurable robot is chain-type robots. These robots are not organized in a lattice structure and therefore a self-reconfiguration step consists of an additional step in which two modules that are about to connect search for each other before connecting. They have to perform this search since they cannot rely on each other being in a certain position, as they would be in a lattice-type robot.

Self-reconfigurable robots have some unique features that make them interesting from an engineering point of view. Owing to their modular nature, self-reconfigurable robots have a high degree of redundancy, which can be exploited to become robust. A hardware failure or a software error may cause a module to fail but does not cause the robot as a whole to fail. The remaining modules can compensate for the loss of a module. Therefore the system is robust and its performance degrades gracefully with the number of failed modules. It may also be possible for the robot to use its ability to change shape to replace broken modules with spare ones in the system if any exist. Through this self-repair ability, the robot may stay functional even if a substantial number of modules fail.
A self-reconfigurable robot is versatile: the modules can be combined in many different ways, allowing them to form the basis for a wide range of different robots. Furthermore, a self-reconfigurable robot is adaptable because it can continually adapt and even completely change shape if a task requires it.

Self-reconfigurable robots are cheap compared with their complexity. The individual modules are quite complex and expensive to produce. But, a modular robot consists of many identical modules and therefore the cost of the individual module can be lowered because they can be mass produced.

In summary, modular self-reconfigurable robots are versatile, adaptable, robust, and cheap compared with their complexity. It is important to note that these features are only potential features. In theory, it should be possible to realize these features based on the concept. However, in practice, the features are often realized only to a limited degree.

![Figure 7.2: An ATRON module](image)

### 7.1.3 ATRON Robot architecture

ATRON modular robots [57] are used for the experiments in this work. ATRON is a homogenous, lattice-based self-reconfigurable modular robot. This module is composed of two half-spheres that can rotate indefinitely relative to each other with the speed of 60 degrees per second. A module has four male and four female connectors, two of each type on each hemisphere. The male connectors consist of two opposed sets of hooks that can grab onto the two bars of the female connectors as shown in Figure 7.2. An ATRON module weighs
0.850 kg and has a diameter of 110mm. A module consists of different sensors such as tilt and infra-red sensors. The ATRON modules can be connected to each other in many ways, forming many different robot morphologies, for example snake and four-legged crawling robots, as shown in Figure 7.3. The robot can change between different morphologies by going through a relatively complex sequence of self-reconfiguration steps.

7.1.4 Robot simulator

Simulation experiments of the ATRON robot are performed in an open-source simulator named Unified Simulator for Self-Reconfigurable Robots (USSR) [23]. The simulator is based on Open Dynamics Engine which provides simulation of collisions and rigid body dynamics. Physical forces like gravity and friction are implemented and the parameters, e.g. strength, speed, weight, etc., has been calibrated with the existing hardware. The physical validity of the mechanical simulation has been demonstrated by using the simulation results in real robot in some experimental works [22].
7.1.5 Control in a Modular Robot

Although an important aspect of modular self-reconfigurable robots is the self-reconfigurability, but these robots most of the time stay in a fixed configuration and perform their relevant task. Only in case of special changes in environment or required task they self-reconfigure to a new configuration proper for the new behavior.

This fact, raises the challenge of how toe control modular robots in fixed configurations to perform specific tasks. The task we have focused on here is locomotion of the modular robots.

![DNA and cell images](image)

**Figure 7.4:** a) Different cells develop from identical genome, b) Different watermelons with the same genetic code, are grown in different environments
7.1.6 Implementing FGRN to control modular robots

Nature makes Robust, Efficient, and Successful biological entities. Biological entities are controlled in a distributed way in cellular level which is similar to the situation in modular robots. As we explained in Chapter 2, the controller of a biological cell is encoded in the genome. Genome is not a direct plan but it works as a blueprint containing instructions of development of the cell. On the other hand, in a biological organism all cells contain an identical genome. Different phenotypes can be produced from these identical genotype in different environmental conditions. As we can see in Figure 7.4, in this way, we can grow different natural products from an identical genetic code.

These ideas can be used in the modular robots. We can consider modules of a modular robot as the cells of a multi-cellular organism. On the other hand FGRN systems can be implemented in distributed way. FGRN genomes are indirect encoding which can drive the behavior of a system. Also, an identical FGRN genotype can be used in different FGRN cells and make different behaviors. In the distributed system of a modular robot, we put a FGRN cell in each module to be used as its controller. All the FGRN cells share the same genotype but they run independently and in parallel to each other (Figure 7.5).

**Figure 7.5:** An identical version of FGRN genome is copied to all the local controllers of the modules.
7.5). By providing different environmental information for the FGRN cells based on the situations of each module, i.e. information about positioning of the module or some sensory information, the FGRN cells might follow different developmental trends and make different output patterns for their modules.
7.2 Fractal Gene Regulatory Networks for Robust Locomotion of Modular Robots

As mentioned in the previous section, FGRN sounds potentially useful in controlling modular robots. This section reports the experiments which are performed in order to investigate this hypothesis in practice. We evolved FGRNs as local controllers of ATRON robots for a locomotion task. In order to keep things as simple as possible we didn’t use any communication between modules and no access to the sensors is implemented for the controllers. The experiments are evaluated based on the speed of the locomotions and the achieved results are compared to previous results from a learning method [22].

Since evolvability of a solution is an advantageous point for a system, some other experiments are also performed to investigate evolvability of the achieved solutions in the case of module failure and it is shown that the FGRN system is capable of coming up with new effective solutions in this case.

![Figure 7.6: Different morphologies of the ATRON robot in the locomotion task.](image)

7.2.1 Problem Description

Three different morphologies of ATRON modular robot (Figure 7.6) are considered to be controlled by FGRN local controllers. The task is locomotion of the robots. FGRNs are evolved to move the robot as fast as possible. No sensor information and communication between modules are used.

Each FGRN controller is supposed to make a control command for its ATRON module in each time step. ATRON modules consist of two hemispheres which can rotate in respect to each other. The control commands are sent to the actuator and are one of the following three commands:

- `rotateRight` - rotate clockwise 90 degrees
- `rotateLeft` - rotate counterclockwise 90 degrees
• *stop* - rotate zero degrees

The FGRN genomes are evolved such that when they are copied into the FGRN cell (controller) of the modules and provide them with some initial appropriate environmental information related to the module, the cell generates a right sequence of control commands for its module and makes the robot as a whole to perform its locomotion task successfully.

### 7.2.2 Computational Procedure

Every module of a robot is considered a cell in a multicellular creature. Each module contains an FGRN cell which includes its genome and cytoplasm. All the FGRN cells run in parallel and independent of each other and make their own sequence of output commands for the modules containing them.

All the cells are genetically identical which means they contain an identical copy of a genome. Environmental information about the number of connections and the initial orientation of the module which contains the cell is provided for each cell in the form of environmental proteins. Therefore, two cells which are contained in two modules with different environmental situations initially contain different proteins in their cytoplasm. Different cytoplasm content might activate different genes of the genome of each cell and leads to different internal interactions and developmental trends. Consequently, while the cells are genetically identical, different phenotypic characteristics might be formed and different output commands might be generated by the cells during their lifelong development.

The run-time procedure of a robot can be summarized as follows:

- Create genome
- For every Module of the robot:
  - Make an empty FGRN cell and put a copy of the genome into it.
    * During the run-time of the robot:
      * Receive information about the module’s environment and activate the relevant environmental proteins.
      * Develop the cell for one cycle according to the developmental steps in section 3.2 and receive cell output.
      * Translate cell output to the module command and
      * Execute the command.
7.2.3 Simulation Results and Discussion

7.2.3.1 Genetic and developmental configurations

A population of 20 FGRN genomes is evolved for 50 generations. Each genome is initialized with randomly generated regulatory, receptor, environmental, and behavioral genes. The initial number of each type of gene and the genetic parameters are shown in Table 7.1. Evolution is allowed to regulate the number of each type of genes.

To evaluate a genome, identical versions of a genome are copied to all the modules’ FGRN cells. Each cell receives some environmental proteins describing the number of connections and the initial orientation of the module in which it is situated. Also an additional environmental protein common between all the cells is initially provided.

In order to make an action for each module in every step, modules independently run their own FGRN cell for one developmental cycle and receive an output from the cell. The cell output is calculated on the basis of activation of behavioral genes inside the cell. The output is mapped to one of the three commands: rotateRight, rotateLeft, stop.

After a specified time span (50 sec.), fitness is evaluated as the distance between the initial position and the end position of center of mass of the robot.

<table>
<thead>
<tr>
<th># population size</th>
<th># generation</th>
<th>crossover rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>40%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mutation rate</th>
<th>#regulatory genes</th>
<th>#receptor genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1%</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#environmental genes</th>
<th>#behavioral genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/10 (for crawler)</td>
<td>1</td>
</tr>
</tbody>
</table>

7.2.3.2 Case studies

We have evolved multicellular FGRN controllers for three robots with different morphologies and the same genetic configurations. Figure 7.6 shows the three morphologies which are used. In these experiments, no sensor is used and there is no communication between the modules.
Figure 7.7: Velocity of the Quadrupedal, Crawler, and the two-Wheeler robots.
As it might be expected, for the two-wheeler robot, evolution leads to controllers which rotate the two opposite modules in the opposite directions to move the robot like a car. For the quadrupedal robot, a swimming-like behavior evolved. For the crawler robot, different crawling gaits evolved. In order to evaluate the robots, the velocity of the locomotion is calculated for each robot. The best and population-average velocities are shown in Figure 7.7. The velocities are calculated as the amount of displacement of the robot in the specified time-span (50 sec). The figure shows the results averaged over 10 independent runs.

This problem has previously been investigated by a learning strategy in [22]. The applied learning strategy is reinforcement learning accelerated by a heuristic which detects and repeats potentially underestimated actions to accelerate the estimation accuracy and presumably accelerates the learning. We compared the results achieved here by the results of the learning approach. Table 7.2 shows the higher velocities achieved by the FGRN controllers and the mentioned learning controllers.

Table 7.2: Comparison of the best velocities achieved by FGRN and [22] learning algorithm.

<table>
<thead>
<tr>
<th>Robot Configuration</th>
<th>Learning Mean</th>
<th>FGRN (Population average) Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadrupedal</td>
<td>0.0208</td>
<td>0.0260</td>
<td>0.0011</td>
</tr>
<tr>
<td>Crawler</td>
<td>0.0210</td>
<td>0.0248</td>
<td>0.0038</td>
</tr>
<tr>
<td>Two-wheeler</td>
<td>0.0383</td>
<td>0.0586</td>
<td>0.0007</td>
</tr>
</tbody>
</table>

**7.2.3.3 Is the FGRN system still evolvable?**

In another experiment, the evolvability of the FGRN system is investigated after a module failure. The crawler robot is selected for this experiment.

We considered the solutions found in the last experiment. Different gaits were recognizable between the solutions evolved in the 10 runs. Based on the position of the modules which had more effect in the locomotion, the solutions can be categorized in two main groups- solutions which mainly use the shoulder modules and solutions which mainly use the arm modules (See Figure 7.6).

The second group which has the velocity of higher than average-velocity is selected. In order to resemble a situation of failure, one of the modules of high importance (one of the arms) is disabled while the robot uses the previously evolved FGRN controller. Since the controller is not suitable for...
Figure 7.8: Velocity of the crawler robot before a module failure, after failure, and after more evolutionary generations.

this new situation, the fitness falls considerably. Afterwards, the controllers are allowed to evolve for 30 generations and the velocities of the new solutions are evaluated.

As it is shown in Figure 7.8, the velocity of robots falls after failure, and then rises when evolution continues. The performance of the new evolved controllers is investigated for robots both with the broken module and intact module (after repairing the broken module). Table 7.3 shows the velocities in different situations and represents a good performance for the new evolved controllers in both cases of intact and broken modules. Furthermore, the experiment repeated with the broken module to evolve controllers from scratch (See Figure 7.8). The velocities are averaged over 10 runs of evolution (Table 7.3).

<table>
<thead>
<tr>
<th>Table 7.3: Averaged velocities of the Crawler robot.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before failure</td>
</tr>
<tr>
<td>All runs 0.0248</td>
</tr>
<tr>
<td>Selected runs 0.0262</td>
</tr>
<tr>
<td>After failure (Selected runs)</td>
</tr>
<tr>
<td>Failure happened 0.0145</td>
</tr>
<tr>
<td>More evolution 0.0245</td>
</tr>
<tr>
<td>Module repaired 0.0235</td>
</tr>
<tr>
<td>Broken module from scratch 0.0244</td>
</tr>
</tbody>
</table>
7.3 Sensor-coupled Fractal Gene Regulatory Networks for Locomotion Control of a Modular Snake Robot

This section further investigates the usefulness of FRGN for control of modular robots; in particular, we extend on previous section by looking at sensor-inputs which are integrated with FGRN in the local controllers of the modules. The FGRNs are evolved here as local controllers of modules of a snake modular robot receiving inputs provided by the local sensors of the modules.

In modular and multi-segment robotics, controllers for snake robots have been extensively studied based on gait control tables [71], Central Pattern Generators (CPGs) [40, 43], artificial hormones [38], and role-based control [66]. However, opposed to our controllers these controllers except [43] are open-loop and in the case of the latter two rely on explicit communication between modules for synchronization.

In the current experiments, there is no communication between the modules and the synchronization of the modules takes place through the feedback received from environment by the means of sensors. The usefulness of this sensor-coupled FGRN controller is studied and the capability of FGRN controllers is observed to come up with the proper solution in case of different levels of the sensory information.

7.3.1 Problem Description

FGRN local controllers with access to tilt-sensor inputs are evolved for a locomotion task of a modular robot. The controller is evolved for a snake-shaped robot consists of seven ATRON modules (See Figure 7.9a). Every module contains three tilt sensors as (TiltX, TiltY, TiltZ). The sensors specify the direction of gravity related to the coordination system of the module (Figure 7.10). The initial tilt sensor values of a module are different for the neighbor modules because of the positioning of the connectors in ATRON. The initial values are (0, -90, 0) for the modules in the odd positions of the snake and (-90, 0, 0) for the ones in the even positions. (Figure 7.9b)

The modules are controlled locally and there is no explicit communication or synchronization between them. So, they can synchronize implicitly using their sensors, and coordination of their behavior takes place through the environment.
First, we evolve the controllers with access to all the three tilt sensors. Then the access of the controller is restricted to one of the sensors that carries less information.

### 7.3.2 Computational Procedure

Evolution searches for FGRN genomes which are used in the local FGRN controllers to solve the locomotion task. To evaluate a genome, an identical version of genome is copied to all the FGRN cells which are situated in the modules.

In every running cycle for a module, it receives the tilt sensor values from its local sensors and sends it to its own FGRN cell. The modules independently run their own FGRN cell for one developmental cycle and receives an output from the cell in order to make a command to be executed by its actuator. After execution of the command, a new cycle starts for the module. Figure 7.11 shows a module’s running cycle.
In order to sense the environment by an FGRN cell, each cell receives tilt sensor values from the module’s sensors. Initially, one input gene is related to each sensor. The level of protein expression of each input gene is determined by the value received from the related sensor. A development cycle is performed for the FGRN and the new concentration level of each protein in the cytoplasm and the output command is specified. The FGRN output is calculated on the basis of activation level of behavioral genes and the real values of the coding region. The output value received from the cell is scaled and used as the absolute position of the module’s actuator which is between -180 to 180 degrees. Modules use the nearest rotation angle to reach the desired absolute position. For a graphical view of the internal interactions of a module see Figure 7.12.

7.3.3 Simulation Results and Discussion

We performed two experiments with the snake modules. In both experiments FGRN local controllers with access to tilt-sensor inputs are evolved for the locomotion task. For each FGRN controller, robot is controlled by the cell for a specific time period (50 sec.) and fitness is simply evaluated as the average speed of locomotion of the robot.

For each evolutionary run, a population of 50 FGRN genomes is evolved for 250 generations. Each genome is initialized with randomly generated regulatory, receptor, environmental, and behavioral genes. Evolution is allowed to regulate the number of each type of genes.
7.3.3.1 Evolving controllers with unrestricted access sensors

In order to study if FGRN controller can gain any benefit from the tilt sensor inputs, we evolved the controllers with access to all the three local sensors. Evolution was free to use all or some of the sensor inputs for the controllers. The evolved controllers were evaluated in the locomotion task and the speed of locomotion was measured as the distance between the initial position and the end position of the center of mass of the robot and used as the fitness value. We repeated the experiment for 10 independent runs. The average speed of the best controllers from the ten runs was 0.0334 m/s (with standard deviation of 0.0032) and all the runs evolved controllers that generated rolling locomotion.

In order to investigate the effects of different sensor values in producing robot behavior, we limited access of the evolved controller to different combi-
Each module contains an FGRN controller that specifies the actuator’s absolute position. The achieved results demonstrated that for 9 runs out of 10, there is no detectable effect for the TiltY and TiltZ sensors. In the only other run, output was produced based on both TiltY and TiltZ sensor values and no use of TiltX detected. This controller had the speed of 0.027 m/s. In the same way as the sensor values, we removed regulatory genes of the evolved FGRN controllers in order to investigate their influence on the controllers’ behavior. The investigation demonstrated that only in one of the evolved solutions, regulatory genes were participating in producing the controllers’ output. No significant difference was observed between the speed of this controller and the rest.

Based on the above investigations, for the eight runs out of the 10 runs, the evolved controllers produced output merely from TiltX sensor value. This means the controller directly maps one input to the output which is a simple controller for this robot.

For a typical evolved FGRN controller, the internal dynamics of the modules are represented in Figure 7.13. As it is demonstrated in the figure, the output values can be simply calculated using a linear equation. We derived the equation from the related input and output data as:

\[ Output = TiltX \times 0.33 - 60.8 \]
7.3.3.2 FGRN with restricted sensor information

In the second experiment, we investigated whether the regulating dynamics of FGRN can make proper output patterns when the instant values of input sensors doesn’t carry enough information. As the results of the last experiment demonstrated, TiltZ sensor has no detectable effect in producing the control outputs. It made us suspect that this sensor doesn’t have enough information for this control task. Therefore, we first tried to evolve a linear equation solely based on TiltZ sensor. We implemented a real-valued genetic algorithm to evolve a population of 50 individuals for 250 generations. The experiment was repeated ten times and we observed that evolution failed to find a proper controller. Then, we evolved FGRN controllers which have only access to TiltZ sensor value to investigate if FGRN can exploit this restricted sensor information.

We repeated the evolutionary process for 10 independent runs and observed different locomotion-types for the best controllers of the different runs. The loco-motion-types are discussed in three groups. The first group consists of the controllers which generate rolling-type locomotion for the robot. In order to study which parts of the network are involved in the control process, we disabled the sensor and each of the regulatory genes one by one. In all cases the controller failed to make proper locomotion. It demonstrates that both
regulatory genes and sensor input are used by the controller. The Internal dynamics of one of the best controllers we achieved in this group is represented in Fig. 8. In order to get an informal impression of the robustness of the controllers in case a module breaks which lead to restarting controller, we randomly chose a module and restarted its controller to the initial state during the robot’s run. We repeated the experiment several times and observed that the robot continues its normal locomotion after a short while.

The second group includes the controllers that make crawling-type locomotion. Investigation of the internal dynamics of the controller demonstrates that these solutions are mainly based on the regulatory genes and doesn’t really exploit the input information. We observed that these robots are not robust against randomly restarting of the controllers.

The third group consists of the controllers which make efficient locomotion once in a while. Benefiting from the robot’s body accidental flips over, these controllers sometimes make fast locomotion, and otherwise they do not produce locomotion. Since evolution only searches for fast controllers and there was no selection pressure towards the robustness and reproducability of the locomotion, the large fitness that these controllers gain by the accidental success is enough to pick them up among the other controllers in the evolutionary process. These controllers are not robust even during normal locomotion. Average and standard deviation of speed reached by the different controller groups are shown in Table 7.4.

**Table 7.4:** Speeds reached by different types of locomotion

<table>
<thead>
<tr>
<th></th>
<th>All</th>
<th>Rolling</th>
<th>Crawling</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average speed</td>
<td>0.0209</td>
<td>0.0248</td>
<td>0.0168</td>
<td>0.0212</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.0076</td>
<td>0.0047</td>
<td>0.0015</td>
<td>0.0112</td>
</tr>
</tbody>
</table>

The inner dynamics of a typical controller is represented in Figure 7.14. The controller is selected from the rolling-type group which demonstrates an efficient and robust behavior. As it is represented in the figure, the TiltZ value is zero for all the modules on the start of the execution. Therefore, there is no difference between the cells of a robot at the beginning and all of them make the same output for their module actuators. Rotating actuators as a result of command execution, changes module’s orientation. This might lead to different TiltZ sensor values for different modules. After a short while of chaotic behaviors, modules start to synchronize and coordinate their behaviors.
through environmental feedback which is received in the form of the sensor values.

![Figure 7.14](image)

**Figure 7.14**: Internal dynamics of the selected controller of the second experiment for the two first modules. Green lines represent the sensor values, black lines represent the output for the actuator absolute position; and the red and brown lines represent the concentration level of the two regulatory proteins.

**Table 7.5**: Phase shift (per period) between the neighbor modules for the typical controller of each experiment

<table>
<thead>
<tr>
<th>Module number</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
</tr>
</thead>
<tbody>
<tr>
<td>first experiment</td>
<td>0</td>
<td>0.21</td>
<td>0.52</td>
<td>0.34</td>
<td>0.34</td>
<td>0.41</td>
<td>0.48</td>
</tr>
<tr>
<td>second experiment</td>
<td>0</td>
<td>0.56</td>
<td>0.37</td>
<td>0.41</td>
<td>0.40</td>
<td>0.36</td>
<td>0.38</td>
</tr>
</tbody>
</table>

**7.3.3.3 Comparison of behaviors from the best evolved controllers of the two experiments**

In order to have an impression of the rolling behavior produced by the best controllers of each of the two experiments, we studied the actuator’s absolute positions for one typical controller evolved in the first experiment and one typical controller from the second one.

As it is demonstrated in Figure 7.15, the module actuators have oscillatory behaviors. For the first experiment, the average period of estimated oscillation of the actuator absolute positions is 6.46 sec (with standard deviation of 0.63).
The estimated phase shifts between the actuator signals of the consecutive modules is represented in Table 3. For the second experiment, the average period of oscillation of the actuator absolute positions is 7.6 sec (with standard deviation of 0.2). The estimated phase shifts of the consecutive modules are represented in Table 7.5. It is interesting to note that the phase difference between neighbor modules is not constant. We suspect this is because modules are subject to different forces and dynamics depending on their position in the snake.

Figure 7.15: Actuators’ absolute positions of all the modules for a typical controller of the experiments with unrestricted (a) and restricted (b) access to the sensor values.
7.4 Conclusion

In this chapter we explored application of FGRN systems to control modular robots. FGRN systems are inspired by natural cells and due to their internal interactions they are able to generate complex output patterns which might be used as control commands. Implementing the FGRNs in multicellular way provides us a distributed controller for modular robots. The local controllers for all modules are encoded identically and run independently.

We performed two different sets of experiments with ATRON modular robots. First, no communication is used between the modules in order to keep the system as simple as possible. We carried out experiments with different morphologies of the ATRON in a locomotion task and reached good performances. Results are compared to the previously reported results of robots employing a reinforcement learning strategy. Furthermore, we investigated the capability of the FGRN system to evolve more in case of a failure and the achieved controllers are evaluated for both intact and broken robots. The results show that the FGRN system is still evolvable to find new solutions for new situations of the robot.

In the second set of experiments, we explored FGRN controllers for a snake-shaped modular robot where only the tilt sensor inputs are available for the controllers. First, we provided the controllers with all the three tilt sensor inputs. The evolved controllers were simple linear equation which exploits only one of the three sensor inputs and ignores the regulatory abilities. In the next step, we restricted the controller’s access to one of the other sensors and tried to evolve new linear equation controllers based on this information. Since evolution couldn’t find the proper controllers, we suspect that the information provided by that sensor is not enough to use such a simple controller.

Then we evolved FGRN controllers with access to this sensor information. The resulting controllers made appropriate oscillatory output patterns to control the modules. Investigating the different parts of the FGRN genome demonstrated that the system exploits both sensor values and regulatory network capabilities to make the proper controller commands. As it might be expected, the generated outputs of the controllers were oscillatory patterns shifted for each module. Furthermore, we performed some preliminary tests towards robustness of the controllers in both cases and observed that the controllers can drive the robot properly in the case of random restarting of the
controllers during locomotion.

All in all, it is demonstrated that FGRN can be evolved to both simple and relatively complex controllers depending on the problem. Furthermore, when the capability of FGRN to make oscillatory patterns is coupled with the sensor information, the controllers show some degree of adaptability. In this way, the identical controllers generate different oscillatory outputs when situated in different modules and may provide some levels of robustness for the whole system. While it has not been verified we think that the idea of using sensor-coupled FGRN controllers for local control and synchronization between segments can be transferable to other modular robots as well.

The controllers we developed here are tied to the physics of the ATRON self-reconfigurable robot and are thus not directly applicable to control of other modular robots such as M-TRAN [54], SuperBot [66], CKBot [72] due to their differences in weight, actuator strength, placement of sensors, etc. However, it is a general problem of all embodied controllers that they rely on the specific physical properties of the robot on which they run. For the same reason, the controllers cannot be directly applied to control of non-reconfigurable snake robots either (see [69] for an overview). However, the idea of a model-free approach relying on sensors for both local control and implicit synchronization between segments may be transferable to other robots as well. More importantly, we expect our development method based on evolution of FGRN can be applied to these systems.
Chapter 8

Conclusion
Conclusion

This thesis has presented several experiments (Chapter 5, 6, 7) to investigate and implement the features and properties of FGRN as a method of developing solutions through evolutionary processes.

In chapter 5, FGRN is employed to successfully solve a non-Markovian motion planning benchmark problem. The FGRN is implemented as the controller of an agent. The problem required a kind of memory. Thus, the FGRN is augmented by input capabilities. This capability along with the network properties of the FGRN, equips the agent with an implicit memory and leads to proper successful motions for the agent. The experiment is repeated for the ordinary FGRN without input abilities. The achieved results are compared with other methods and remarkably enhanced results are observed for the proposed method. The effects of the input capability and the behavior of the developmental solutions are experimentally investigated and interesting points of the solutions are highlighted.

In chapter 6, a population of FGRNs augmented with input genes and capable of sensing the fittest already generated pattern is considered for solving a long pattern generation problem. The idea was to let the FGRN to sense the best solution from previous generations during its developmental steps and observe the evolutionary behavior of the population and the degree of participation of different types of genes during generations. The achieved results demonstrated that when enough generations are passed the behavior of an individual becomes more dependent on input genes than the regulatory genes and it leads to an imitational behavior. Individuals in the evolving population become increasingly good imitators but they still can gently modify the top pattern of the society. The proposed method is compared to an ordinary FGRN and higher performance is detected for the method.

In chapter 7, FGRN is employed for distributed control of modular robots. FGRN is implemented in a multicellular way where several FGRNs initiated from an identical genome work independently to control several modules of a robot.

Experiments are performed with different morphologies of a modular robot in a locomotion task and good performance is reached comparing a previously applied learning method. The capability of the FGRN system to evolve more in case of a failure is evaluated and the achieved controllers are evaluated for both intact and broken robots and it is demonstrated that the FGRN system
is still evolvable to find new solutions for new situations that the robot may encounter.

In another set of experiments, input-coupled multicellular FGRN is implemented to control a modular robot. The FGRN were evolved with access to three sensors of the modules and the evolved successful controllers were surprisingly simple. Then the access of the FGRNs were restricted to one of the sensors carrying less instant information and FGRN were still evolvable to find the proper controllers for the robot. It is demonstrated that FGRN can be evolved to both simple and relatively complex controllers depending on the problem.

All in all, as a computational GRN, FGRN demonstrated the capability of combining input information with its intrinsic oscillatory generation abilities. It also presented a good evolvability in producing appropriate outputs in order to solve different problems. The system also shows interesting properties and enough potential for extension in different areas. Since the method is model-free and problem-independent, we suspect that the system can be useful for evolving complex solutions with a minimum modification for different problems. The system can be applied to many other problems in the area of modular robotics, for example in self-reconfiguration of the robots. The system can be evolved while the difficulty of the problem gradually increases. It may also be considered to evolve reusable genetic codes. For example it can be evolved in some special cases and parts of the evolved genetic code (a number of genes) can be used to evolve more complex solutions faster. Furthermore, the behavior of the system can be studied in various circumstances specially in artificial life type situations. For example, the evolutionary behavior of the population can be studied when the environment changes in different frequencies of producing special proteins during development of the system. As another direction of study, the system can be analyzed from network point of view. It can be represented in new forms in order to facilitate understanding and analyzability of the evolved solutions and the properties of the system can be investigated more.
Appendix A

Mandelbrot Fractal Set

A.1 Introduction

Fractal sets were known for a long time in mathematics. But before 1970’s they were not popularized and not be used in practice. In 1970’s a mathematician Benoit Mandelbrot coined the word ”fractal” and attracted a lot of interest towards geometry of fractals. He applied fractal geometry to different areas such as physics and biology. He wrote his famous book *The fractal Geometry of Nature* [53] and showed that several natural phenomena can be well described by fractal geometry. Fractals have the property of self-similarity and can be found everywhere in the nature. Figure A.1 shows some examples.

A.2 Fractals in nature

Design and structure of the nature has always been an interesting subject for scientists. Up to the 20th century, Euclidean geometry sounded to be the final answer to the question of how to describe phenomena in the nature. The geometry were dealing with the wholes rather than fractional realities. Plane geometry concerns planar (one- and two- dimensional) structures, and solid geometry describes volumetric (three-dimensional) structures.

This approach is fine until some level of complexity and with some fundamental suppositions but what if we change the suppositions. Suppose the number-expressing dimension can be a fraction. Suppose shapes are defined by iterating an equation instead of solving an equation once. Benoit Mandelbrot made a number of the suppositions, and made the fractal (fractional) geom-

\[\text{images are from wikipedia.org}\]
Geometry and mathematics. He was interested in irregular and seemingly chaotic patterns. Phenomena such as frequency of earthquakes, flooding conditions, changes in prices during long periods which all seemed to occur with a regular irregularity. These phenomena all had one important property in common: they all have self-similarity. Self-similarity is symmetry across scale. It implies recursion, pattern inside of pattern.

Trees and coastlines are some well-known examples of fractals in nature. A photograph of a section of coastline from a helicopter will show the same ragged contours as a photograph of the whole coast taken from a space station. A photograph of a one-meter section of the same coast will also show the same contours. The various coastlines are self-similar, each like the others in shape, but different in magnitude [37].

Fractal designs are also recognizable in many biological structures and processes in our bodies. The vascular tree and airway of the cardiac conduction system and neurons, and structure of biological interfaces such as cytoplasmic membrane, plasma membrane, gas exchange surface of the lung are some examples. It sounds that nature build things in different scales instead of only one design for each specific thing. For example similar features can be found among species with different sizes from microbes to the elephants [56].
Figure A.1: Fractals can be found everywhere in nature. a) Fern; b) A cross between broccoli and cauliflower; c) Growth pattern on a leaf; d) Chaotic branching pattern from electrical discharge; e) Rivers in Savannah Georgia; f) Natural cloud spiral; g) Bacterial colony; h) Another bacterial colony
A.3 Mandelbrot fractal set

Mandelbrot fractal set is a fractal set named after Benoit Mandelbrot and popularized the Mandelbrot fractal set. A fractal displays self-similarity at various locations, and various scales. The patterns are the same exactly, approximately or even stochastically. Magnifying a fractal reveals small-scale details similar to the large-scale characteristics. The Mandelbrot set is self-similar at various scales, but the details in small-scale are not exactly identical to the whole. Although the set is infinitely complex, generating it is based on an extremely simple equation in complex numbers. The equation is as follows:

\[ Z = Z^2 + C \]  

(A.1)

A.4 Generating the Mandelbrot fractal

The Mandelbrot set is a set of complex numbers, so we plot it on the complex number plane. In order to find the numbers which are part of the set we use a test. The test is based on the equation \( Z = Z^2 + C \).

\( C \) is the number we are testing and is constant during the testing process. If \( C \) shows up to be in the set, it will be plotted in the complex plane. \( Z \) starts out in zero and changes as we repeatedly iterate the fractal equation. With each iteration we create a new \( Z \) that is equal to the old \( Z \) squared plus the constant \( C \). So the number \( Z \) keeps changing during the test.

We are just interested in the magnitude of \( Z \). The magnitude of a complex number is its distance from zero in the complex plane. To calculate it, we add the square of the real part of the number to the square of the imaginary part of it and take the square root of the result.

As we iterate the equation, \( Z \) changes and the magnitude of \( Z \) also changes. For different values of \( C \), this magnitude either stays equal to or below 2 forever, or it will eventually surpass 2. Once the magnitude of \( Z \) surpasses 2, it will increase forever (See Figure A.2). If the magnitude of \( Z \) stays equal to or below 2, \( C \) is part of the Mandelbrot set and should be plotted. If the magnitude of \( Z \) eventually surpasses 2, the number is not part of the Mandelbrot set.

In order to graph a nice image of the fractal, we can test the numbers in the complex plane with a high resolution. We can also add color to the image or show it in gray-scale to enhance it aesthetically and show more information about the points in the image (Figure A.3). The colors are added to the points
that are outside the set, according to how many iteration were required before the magnitude of Z surpassed two. Note that in practice we have to stop the test for each number after a maximum iterations if the number is still below 2. So, the number of colors or gray-scales we need is limited to the maximum number of possible iterations we set.
Figure A.3: Mandelbrot fractal set.
References


چکیده

تكامل و گسترش در شبکه‌های تنظیم زنی

به کوشش:

پیام زهادت

شبکه‌های تنظیم زنی، بخش اصلی فرآیندهای زیستی در طبیعت هستند. در حالی که زنون یک سول‌زنه، الگوی ساخت موجود زنده را در خود دارد، گسترش ساختار و رفتار سیستم از طریق تراکنش-های مداوم بین ژن‌ها و پروتئین‌هایی که توسط ژن‌ها کد شده‌اند، صورت می‌گیرد.

شبکه‌های تنظیم زنی مرتبط به فرآیندهای زیستی، در طول فرآیند طولانی تکامل شکل گرفته‌اند و رفتارهای پیچیده موجودات زنده طبیعی را کنترل می‌کنند. در دنبال پردازش تکاملی، پژوهشگران در جستجوی روشهای و شیوه‌های تفاوتی بین بیشترین یافته‌آمیزی تکامل سیستمی به منظور حل مسایل پیچیده هستند. شبکه‌های تنظیم زنی محاسباتی، برای کمک به درک سازوکار موجود در شبکه‌های تنظیم زنی زیستی و همچنین برای ایجاد راه‌حل‌های بی‌پیچیده بر اساس کدهای زنی نسبتا ساده در حل مسایل مهندسی به کار می‌روند. در این پایان‌نامه، چندین مدل مختلف شبکه تنظیم زنی محاسباتی از زوایای مختلف بررسی شده و مفاهیم و کاربردهای یک مدل شبکه تنظیم زنی به‌خصوص به نام شبکه تنظیم زنی فراکتالی، مورد بحث قرار گرفته است. مدل اصلی جهت ایجاد راه حل‌های مناسب و مطالعه سیستم در شرایط مختلف وبرای پیش‌بینی یافته‌های است. آزمون‌های گوناگونی انجام شده و کاربردهای مدل بسیار مفاهیمی بررسی شده است. نتایج به‌دست آمده، در طول پایان‌نامه ارائه شده و جزئیات آن مورد بحث قرار گرفته‌ند.
پایان نامه دکتری در رشته مهندسی کامپیوتر (هوش مصنوعی)

تکامل و گسترش در شبکه‌های تنظیم زنی

بهوسیله:
پیام زهادت

استاد راهنما
دکتر سراج الدين کاتبی

بهمن ماه ۱۳۸۹