Mathematical Models of Evolvability

by

Shruti Kaushal
2016MAS7069

A thesis submitted in partial fulfillment for the degree of
Master of Science

to the

Department of Mathematics
Indian Institute of Technology Delhi

August 2018

Certificate

This is to certify that the thesis titled,"**Mathematical Models of Evolvability**" is a bona fide record of work done at Indian Institute of Technology, Delhi by **Ms Shruti Kaushal**, for the award of the degree of **Master of Science** in **Mathematics**. The work in this thesis was conducted under our supervision and guidance. The results contained in the thesis have not been submitted elsewhere, wholly or in part for the award of any other degree.

Dr. Niladri Chatterjee
Professor
Department of Mathematics
Indian Institute of Technology Delhi
Hauz Khas, New Delhi -110016, India

Dr. Sumeet Agarwal
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology Delhi
Hauz Khas, New Delhi -110016, India

INDIAN INSTITUTE OF TECHNOLOGY DELHI

Abstract

Department of Mathematics

Master's Thesis

by Shruti Kaushal

Evolution is one of the most fundamental concepts of biology that tries to explain the inherent complexity of organisms. Even though evolution has inspired many computational algorithms, like genetic algorithms, it was only in the past decade that the idea of evolution being a form of constrained learning was proposed by Valiant [1]. Valiant shows that the class of parity functions is not evolvable but linear conjunctions and disjunctions are. Parity functions are Boolean functions which output a true value $(+1)$ if odd(even) number of inputs or literals are true otherwise false (-1). If evolution is a form of learning then by computational learning theory there have to be limitations to what can and cannot be evolved. But this seems counter-intuitive as the most widely accepted theory of evolution, Darwinism, states that the present stage of complexity of all organisms is the result of accumulation of small changes over thousands of generations. Also, parity functions are expressed in various biological networks like the exclusive-OR function in real signal transduction networks. Now, the question is that if there are limitations to evolvabilty how do we explain the existence of something in nature that is deemed to be not evolvable. Valiant seeks to account for these limitations of 'evolvability' in terms of a structural property known as modularity. Systems having different functional complexes, with identifiable separate roles, that work independently and have very few connections among themselves are said to be modular. Valiant conjectured that modularity might enhance the evolvability of functions which are proven to fall under the class of non-evolvable functions. Such complex functions or traits could be learnt as a cumulative effect of several modules responsible for expressing different evolvable functions. Incidentally, modular structures are found in abundance in various biological networks like signal cascading. Our work seeks to characterize the relationship between modularity and evolvability, especially for non-linear functions.

We study different mathematical models of regulatory networks and evolutionary algorithms. Friedlander *et al.* conducted *in silico* simulations of evolution using a mathematical model for gene regulatory networks and found that the type of mutation affects the modularity of the evolved population. We study different factors that could potentially affect the modularity of the evolved population by conducting simulations of the genetic algorithm and varying model parameters like mutation rate and size.

We combine ideas from Valiant's framework and simulation-based approaches to evolvability to study the question relating to the limitations of evolvability posed above. We develop variants

of the Friedlander *et al.* evolutionary algorithm, bringing in ideas from Valiant such as looking at binary-valued inputs/outputs and accordingly adjusting the fitness metric. One such variant can evolve non-linear functions such as exclusive-OR, in a fashion analogous to artificial neural networks. We conduct extensive simulations to study how the time complexity of evolving a particular kind of target function varies with model parameters. We show how regulatory models with multiple stages of computation between input and output (such as signal transduction cascades) are essentially equivalent to neural network or deep learning models. Hence, by introducing non-linearity in the intermediate layer of computation in the Friedlander *et al.* model, we can evolve parity functions using their evolutionary algorithm. However, the time complexity of evolving these grows very rapidly with the number of inputs, in line with Valiant's result of the non-evolvability of such functions in polynomial time. So our analysis suggests that nonlinearity in combinatorial regulation may only be feasibly evolvable for a small number of inputs, given reasonable constraints on the evolutionary algorithm; but that the nature of such evolvability can be understood in similar fashion to non-linear regression via neural networks.

Acknowledgements

I would like to express gratitude towards my supervisors Professor Sumeet Agarwal and Professor Niladri Chatterjee for giving me an opportunity to work on this project. Their thought provoking questions and suggestions were instrumental in shaping my work. They have been very sensitive towards my health problems and I thank them for affording me the flexibility to progress at my own pace. I wouldn't have succeeded if it wasn't for their constant support and guidance.

I would also like to thank my parents for their unconditional love and support. I owe all my achievements to them. To my sister Shradha, I feel a deep sense of love and appreciation. She helped me up whenever I was down and dealt patiently with my anxiety. She has always been a source of emotional strength, happiness and warmth. I dedicate this work to my parents and my sister.

Lastly, I would like to thank my friends Dashmeet Singh, Pooja Gupta and Urvashi Bhatia for standing by me through thick and thin. I express my gratitude to Dashmeet Singh who made my journey at IIT easy and believed in me even when I doubted myself.

This thesis would not have been possible without the support and love from my family and friends.

Shruti Kaushal
2016MAS7069

Contents

Certificate i

Abstract ii

Acknowledgements iv

List of Figures vii

List of Tables viii

Abbreviations ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Definition . . . 1
- 1.3 Organization of Thesis . . . 2

2 Literature Review 3
- 2.1 Wagner's Theory . . . 3
- 2.2 Valiant's Framework . . . 5
  - 2.2.1 Model . . . 6
  - 2.2.2 Performance Metrics . . . 6
  - 2.2.3 Evolvability . . . 7
- 2.3 Alon's Framework . . . 9
  - 2.3.1 Model . . . 9
- 2.4 Conclusion . . . 11

3 Simulations of Network Evolution 12
- 3.1 Modularity Metric and Factors Affecting Evolution of Modularity . . . 12
- 3.2 Simulations . . . 13
  - 3.2.1 Initialization . . . 13
  - 3.2.2 Results . . . 14
  - 3.2.3 Dependence on Modularity Metric . . . 17
- 3.3 Conclusion . . . 18

4 Amalgamation of Simulation and Theoretical Frameworks for Evolvability 19
- 4.1 Introduction . . . 19
  - 4.1.1 Correspondences . . . 19
  - 4.1.2 Newman-Girvan Modularity . . . 20
- 4.2 Function Definitions . . . 20
  - 4.2.1 Single Target Functions . . . 21

4.2.1.1 Importance of a Dummy Variable . . . 22
4.2.2 Multi-Target Functions . . . 23
4.2.2.1 Simulations . . . 25
4.3 Visualization as Neural Networks . . . 26
4.3.1 Significance . . . 28
4.4 Conclusion . . . 28

5 Modifying the Genetic Algorithm to Study Non-Linearity 29
5.1 Remodelling the Mutation Space and Mutation Algorithm . . . 29
5.1.1 Mutation Schemes Analogous to Product-Rule and Sum-Rule Mutations . 30
5.1.1.1 Multiplicative Mutation . . . 30
5.1.1.2 Additive Mutation . . . 31
5.2 Evolving Multi-Target Monotonic Conjunctions and Disjunctions . . . 32
5.2.1 Solution Space . . . 32
5.2.2 Simulations . . . 34
5.3 Evolving Non-linear Boolean Functions . . . 36
5.3.1 Addition of Non-Linearity . . . 36
5.3.2 Parity Functions . . . 36
5.3.3 Simulations . . . 37
5.3.3.1 Initialization . . . 38
5.3.3.2 Results . . . 38
5.4 Conclusion . . . 39

6 Conclusion 41

Bibliography 43

List of Figures

2.1 An example of a modular genotype-phenotype mapping (Figure from [2]). 4
2.2 Effect of different types of natural selection on trait distribution. (Graphs taken from `http://www.sparknotes.com`) 5
2.3 Valiant's algorithm for learning a function 9
2.4 Evolutionary Algorithm (Figure from [3]) 10

3.1 Comparison of fitness of populations evolving through sum-rule and product-rule mutations 14
3.2 Comparison of modularity of populations evolving through sum-rule and product-rule mutations 14
3.3 Comparison of number of zeros in a population evolving through sum-rule and product-rule mutations 15
3.4 Number of zeros in matrix population A and B evolving through product-rule mutation 15
3.5 Number of zeros in matrix population A and B evolving through sum-rule mutation 15
3.6 Probability density function of $\mathcal{N}(1, 0.27)$ 16
3.7 Comparison of all three versions of the algorithm for calculating modularity . . . 17

4.1 Average number of generations the algorithm takes to converge for every mutation size (σ) 24
4.2 Average performance of the population while evolving towards the ideal function (Equation 4.8) through sum-rule and product-rule mutations. 25
4.3 Modularity of the population while evolving towards the ideal function (Equation 4.8) through sum-rule and product-rule mutations. 26
4.4 Neural network visualization of a single target function 27
4.5 Neural network visualization of a multi-target function 27

5.1 Average number of generations the algorithm converged in to multi-target monotonic conjunctions 35
5.2 Percentage of converging runs in evolution of multi-target monotonic conjunctions 35
5.3 Percentage of Converging Runs for Parity functions and Conjunctions with n inputs evolved using **multiplicative mutation** 38
5.4 Average number of generations the algorithm converges in to parity functions and conjunctions with n inputs using **multiplicative mutation** 39

List of Tables

4.1 Correspondences of variables defined in Valiant's and Alon's mathematical setup 20
4.2 Case $1 : w_1 > w_2 > 0$ 23
4.3 Case $2 : 0 > w_1 > w_2$ 23

Abbreviations

TF	**T**ranscription **F**actor
NN	**N**eural **N**etwork
XOR	E**x**clusive **or**

Chapter 1

Introduction

1.1 Motivation

The origin of modularity is a well debated topic but what we can say for sure is that modular structures become increasingly common as we move up the hierarchy of life. It is seen that complex organisms have a high level of differentiation which arises due to the presence of functional complexes in the form of well-separated modular structures. But this type of internal organization becomes extremely rare with rudimentary organisms. Since, every organism has reached a level of complexity through evolution over several generations it seems intuitive to think that modularity too was a property that was acquired over the generations. Since, complex organisms are higher up the hierarchy of life, they perform better in terms of fitness than the more simpler organisms.

Is there a direct or an indirect relation between modularity and fitness? Is modularity desirable and if so, why? Why is that modularity is in seen only in complex organism? It is possible that modularity enhances evolvability i.e. some traits would be in-acquirable without the presence of modular structures or modularity is essential to overcome some kind of limitations faced by the organism with its present level of organization? If so, then what are these functional classes?

This research work is motivated towards answering all the above questions using mathematical tools and concepts from machine learning.

1.2 Problem Definition

This thesis is guided towards analyzing the relationship between modularity and evolvability in a mathematical setup wherein evolution is regarded as a constrained form of learning. We are also concerned with the origin of modularity and what factors play a rule in favoring such structures. We base our research on the theory given by Wagner [2] that modularity seems to be an evolved property rather than being an intrinsic trait and conduct simulations of the genetic algorithm proposed in Friedlander *et al.* to characterize situations that lead to modular solutions. For

establishing a relationship between modularity and evolvability we study non-linear functions and try to find out the role played by modular structures in enhancing evolvability.

1.3 Organization of Thesis

The present work has been divided into six chapters including this the first. Chapter 2 gives a detailed explanation of theories [2], mathematical models given by Valiant [1] and Alon [3] and genetic algorithms studied during the literature review that are crucial to this research. It ends with a discussion on formulating a mathematical model for answering questions about modularity posed in section 1.2 and verifying Valiant's claim about the origin of modularity. Chapter 3 begins with introducing the details of of the mathematical model for gene regulatory networks studied in [3]. Simulations of the genetic algorithm were conducted and the results are carefully documented. A thorough analysis of the results is done which concludes that our results are expected in literature. Different ways of calculating modularity are introduced and the dependence of the results on these metrics is studied. In the next chapter, chapter 4, a variant of the evolutionary algorithm (given by [3] and studied in chapter 3) is proposed, bringing in ideas from Valiant [1] such as looking at binary-valued inputs/outputs and accordingly adjusting the fitness metric. Single-target and multi-target functions are studied. The results from simulating evolution of these functions are studied and certain changes in the evolutionary algorithm are proposed. The chapter ends with visualizing the variant model as a neural network and discussing its significance. Chapter 5 introduces all the changes and modifications proposed in the previous chapter in detail. The mutation space, mutation scheme and the solution space are modified. An analogy between the redefined mutation scheme and the mutation rules discussed in [3] is drawn. The results of the simulations for evolving multi-target monotonic conjunctions and disjunctions are also discussed. The chapter later on introduces non-linearity in the variant model proposed in Chapter 4 which allows for simulating evolution of parity functions. It is shown that although parity functions as a class of functions are not evolvable but with small number of input variables they can be evolved using the modified evolutionary algorithm. The chapter is concluded with a discussion on the significance of a two-layer model and Valiant's claim. The last chapter, chapter 6 concludes this thesis. It briefly summarizes the work of this thesis and concludes and ends with a discussion of possible future directions for research.

Chapter 2

Literature Review

Many papers were reviewed before narrowing in on the problem statement. Three of which form the pith of this research. One of them is *Complex Adaptation and the Evolution of Evolvability* authored by *Wagner* and *Altenberg* [2]. This paper gives a philosophical view of evolution. One of the theories presented in this paper that proved essential in shaping this research was about modularity.
The other two papers - *Evolvability* authored by *Leslie G. Valiant* [1] and *Mutation Rules and the Evolution of Sparseness and Modularity in Biological Systems* authored by *Uri Alon* [3] had an analytic approach in understanding evolution and both presented a mathematical model for the same. The mathematical model formulated by Valiant was very different from the one given by Alon. To understand the results generated in this research one needs to have a thorough understanding of both the models.

2.1 Wagner's Theory

Wagner and Altenberg compare the implications of evolutionary biology and evolutionary computer science in their paper [2]. However, the main reason why this paper became essential to this research was not because of the comparisons it drew between two fields which were supposed to be largely disconnected but for the theory they proposed on how modularity might have come into existence. Wagner described modularity as follows -

> "*... By modularity we mean a genotype-phenotype map in which there are few pleiotropic*[1] *effects among characters serving different functions, with pleiotropic effects falling mainly among characters that are a part of a single functional complex.*"

Modularity can be thought of as a property or a trait of an system having different functional complexes, with identiable separate roles, that work independently and have very few connections among themselves. In this case, the system being studied is the genotype-phenotype maps.

[1] Pleiotropy is the phenomenon of production of two or more apparently unrelated effects by a single gene

A simple illustration (Figure 2.1) given by Wagner explains the concept of modularity of the genotype-phenotype mapping functions.

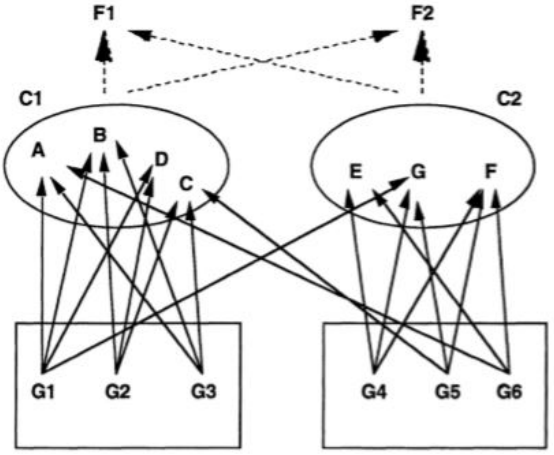
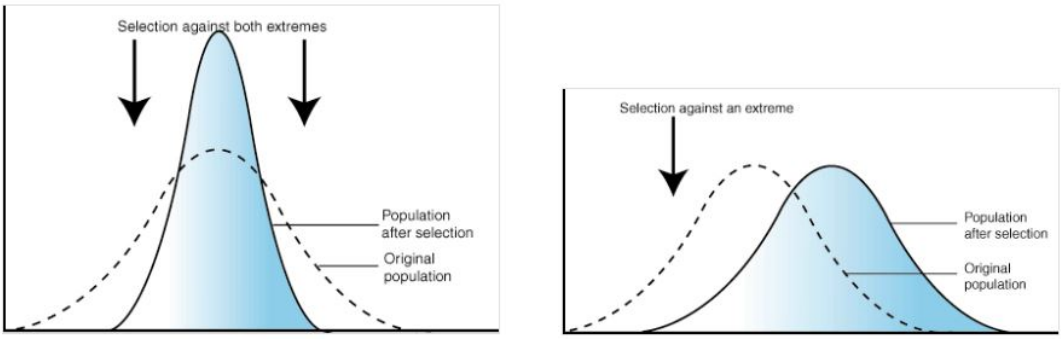


FIGURE 2.1: An example of a modular genotype-phenotype mapping (Figure from [2]).

The above figure is an illustration of a toy example of a genotype-phenotype map. The modular representation of consisting of two character complexes $C1 = \{A, B, C, D\}$ and $C2 = \{E, F, G\}$ responsible for functions $F1$ and $F2$ respectively. $F1$ is the primary function of character complex $C1$ and similarly, $F2$ for $C2$. As it can be seen from the dotted lines, $C2$ has a very weak effect on $F1$ and vice versa. $G1, G2, G3, G4, G5$ and $G6$ represent the genes and their effects are depicted by arrows. Clearly all genes produce more than one distinct effects (pleiotropic effects). Although, pleiotropic effects of genes $M1 = \{G1, G2, G3\}$ are primarily on $C1$ and that of genes $M2 = \{G4, G5, G6\}$ are on $C2$. The representation is modular because there are lesser pleiotropic effects across complexes than within them.

Wagner and Altenberg believed that modularity might lead to enhancement of evolvability; the theory on which this research is based. Since modular systems have different modules responsible for different functions any adaptive change in a function boils down to or is focused at genetically reprogramming the module responsible for the function without affecting any other module. Hence, adaptation of one function is almost mutually exclusive of the adaptation of any other function. Having recognized the importance of modularity, the next step was to question its origin and how did it come into being? Wagner and Altenberg have tried giving a plausible explanation of the same but nothing has been proved as yet. They suggested that although modularity in some cases is an intrinsic property of a mechanism of an organismal function (like enzyme activity), it essentially appears to be an evolved property. Contrary to popular belief that modular structures are formed from integration of smaller sub-units, Wagner believed that rather than integration of sub-units it was the separation of larger units into smaller functional units, also known as parcellation, that may have contributed in the construction of modular structures. Having established that modularity was more of a result of parcellation than integration the next step was to determine how natural selection acted in its favour. Wagner suggested that two types of natural selection - stabilizing selection and directional selection working together might be responsible for causing parcellation.

Stabilizing selection is one of the most long lasting forms of selection acting on species. It favours



(a) Effect of stabilizing selection on trait distribution

(b) Effect of directional selection on trait distribution

FIGURE 2.2: Effect of different types of natural selection on trait distribution. (Graphs taken from `http://www.sparknotes.com`)

the intermediate form of a trait and suppresses the extremes. This leads to a decrease in the variation of a trait. So, through the course of time the extremes are lost and the intermediate form becomes the adaptive trait.
Directional selection is in some way similar to stabilizing selection. It favours one extreme of the trait and selects against the other extreme. This leads to a shift in the trait distribution of the population towards the extreme that is favoured. Figures 2.2(a) and 2.2(b) are an accurate representations of effects of both these selections.

Wagner conjectured that directional selection together with stabilizing selection tend to suppress those pleiotropic effects that possibly connect different character complexes. He further described the process as follows -

"*One possibility of sufficient generality is that the combination of directional selection and stabilizing selection leads to the* ***differential suppression*** *of pleiotropic effects. . . . It implies that directional selection on adaptively challenged character complexes occurs simultaneously selection on all other characters.*"

In conclusion, Wagner and Altenberg believed that modularity was not an intrinsic trait but was in fact selected upon by forces of natural selection and thus can be thought of as a result of evolution. Their theory about the possibility of modular structures improving or enhancing evolvability was crucial in forming the basis of this research.

2.2 Valiant's Framework

It has been widely accepted that the process of learning and evolution have some similarities. Both are adaptive process that can continue indefinitely without any support from an external source or force once set in motion. Leslie G. Valiant was the first to realize these similarities in mathematical terms. He proposed that evolution is a constrained form of computational learning. In his paper "*Evolvability*", Valiant gave a comprehensive mathematical model of evolution and a quantitative analysis of what mechanisms can evolve in a real time with a realistic population size.

Understanding the model formulated by Valiant is important for interpreting and understanding the results generated in the course of this research.

2.2.1 Model

According to Darwinism organisms having more number of offspring have a better chance of survival than the rest (i.e. "Survival of the fittest"). Hence it is crucial to understand how an organism ensures its survival while responding to different conditions and situations.
Valiant treats Darwinian evolution as a form of constrained learning and models it as an optimization problem. He realized the set of all external and internal conditions or experiences of an organism in the form of variables $X = \{x_1, x_2, x_3, \ldots, x_n\}$. For simplicity, the condition variables are taken to be Boolean i.e., $X = \{0, 1\}^n$.
The response function of an organism is modeled as an input-output mapping ($r(X)$). It takes condition variables and their different combinations as input and outputs the response of the organism given by $r(x_1, x_2, \ldots, x_n)$. These functions form the hypothesis for the task of learning an ideal function. Ideal function ($f(X)$) outputs the ideal responses or responses that are desirable for an organism given various combinations of condition variables i.e., $f(x_1, x_2, \ldots, x_n)$. Again, for simplicity, both hypothesis functions and ideal function are taken to be Boolean functions. Valiant studies three classes of functions - Conjunctions, Disjunctions and Parity functions. He proves that conjunctions and disjunctions are learnable in his setup while parity functions are not.
Clearly X is not an exhaustive set of conditions faced by an organism. Indeed, it is virtually impossible to create such an exhaustive set but in the current setup, given that number of condition variables are n , an exhaustive representation of all conditions and their various combinations can only be given by the power set. Valiant denotes this set of all 2^n combinations of values that variables $x_1, x_2, \ldots, x_n$ can take by X_n. The probability distribution over X_n is denoted by D_n. It gives the relative probabilities of various combinations of variable values for $x_1, x_2, \ldots, x_n$ occurring in the context of the organism. Valiant takes D_n to be uniform distribution for simplicity. Also, C denotes a function class.
Before we go into the technicalities of the algorithm, it is important to understand the biological relevance of these functions and function classes. One example can be that the value of x_i gives the expression level of a protein i.e., if $x_i = 1$ then the protein is expressed otherwise not ($x_i = -1$). Similarly, if $f(x) = 1$, where $x \in X_n$, then a further protein is being expressed otherwise not ($f(x) = -1$).

2.2.2 Performance Metrics

All the definitions given below have been taken from [1].
For every optimization problem the objective function and selection criterion need to be defined. In this model, the objective function that needs to be maximized is given by 'performance' defined below.

- **Performance :** The performance of function $r : X_n \to \{-1, 1\}$ with respect to ideal function $f : X_n \to \{-1, 1\}$ for probability distribution D_n over X_n is

$$Perf_f(r, D_n) = \sum_{x \in X_n} f(x)r(x)D_n(x) \tag{2.1}$$

  The performance is a correlation measure between the ideal function f and the given hypothesis function r. Note that the formula is such that it penalizes any point, with non-zero probability, at which the functional value of the ideal function and the hypothesis function differs. The range of performance is $[-1, 1] \subset \mathbb{R}$. The benefits (when both functions agree at a point) and penalties are added up over the set of all experiences to calculate the performance. The organisms that have a high performance are selected preferentially over organisms with a lower performance. This performance measure over the distribution D_n can be viewed as a tness landscape which has to be traversed in order to get as close as possible to the ideal function ($f(x_1, x_2, \ldots, x_n)$) i.e. learn a response function which agrees with f on as many points (conditions) as possible.

  Since the size of the set X_n increases exponentially with an increase in n, it is difficult to calculate the performance for every hypothesis function. So, to reduce the computational complexity, empirical performance is calculated, instead of performance, by sampling only a few experiences from X_n which could be common to all or some organisms.

- **Empirical Performance :** Let $Y \subseteq X_n$ be a limited sampled set of size $s(n)$ of inputs from D_n. Sampling is independent of various r and $s(n)$ upper bounds the population size. So, for a positive integer s, ideal function $f : X_n \to \{-1, -1\}$ and probability distribution D_n over X_n the **empirical performance** of function $r : X_n \to \{-1, -1\}$ is a random variable that makes s selections independently with replacement according to D_n and for the multiset Y so obtained takes value

$$Perf_f(r, D_n, s) = s^{-1} \sum_{x \in Y} f(x)r(x) \tag{2.2}$$

The genetic algorithm for learning the ideal function f involves a mutational process followed by a selection process. Valiant realizes mutation as a random variable (M), in this probabilistic setup, which on an input of a response function r_1 outputs another response function r_2 by conducting a search in a neighbourhood of r_1 subject to some selection criterion based on the functions' performance values. Once this whole setup is established he introduces the concept of evolvability.

2.2.3 Evolvability

Evolvability is an abstract concept that can be visualized as a property of a class of functions. It is very closely related to the definition of learnability and can be thought of as a form of learning with some extra conditions or constraints.
A class C of ideal functions f is evolvable if any f in class C satisfies the following two conditions:

1. For any starting function r_0 the sequence of functions $r_0 \Rightarrow r_1 \Rightarrow r_2 \Rightarrow r_3 \cdots$ will be such that r_i will follow from r_{i-1} as a result of a single step mutation and selection in a moderate size population.

2. After a moderate number i of steps r_i will have performance value significantly higher than the performance of r_0, so that this is detectable after a moderate number of experiences.

These conditions will be sufficient to start from any function and progress towards f, predictably and inexorably. The evolvability of any class of functions depends upon the distribution taken by the condition variables as well as the form of mutation.
Evolvability can be expressed mathematically in terms of performance given by 2.1 and an evolutionary algorithm A. An evolutionary algorithm is defined as a quadruple $A = (R, Neigh, \mu, t)$ where,

- R represents the class of all response functions r.
- Neigh of a given response function r_0 is the neighbourhood of r_0 i.e. the set of response functions into which r_0 randomly mutates.
- $\mu(r_0, r_1, \epsilon)$ gives the probability with which r_0 mutates to r_1.
- $t(\epsilon)$ is the tolerance of the evolutionary algorithm A. It is used to characterize a mutation as beneficial or neutral.

A class of functions C is said to be evolvable if polynomials $s(n, \frac{1}{\epsilon})$ of number of experiences and $g(n, \frac{1}{\epsilon})$ of number of generations, such that for every $f \in C$, $r_0 \in R$ and for every $\epsilon > 0$ with probability at least $1 - \epsilon$, a sequence $r_0, r_1, r_2, \ldots$, where

$$r_i = Mu(f, D, A, r_{i-1}, \epsilon, s(n, \frac{1}{\epsilon}))$$

will have performance

$$Perf_f(r_{g(n,\frac{1}{\epsilon})}, D) > 1 - \epsilon$$

where D is the probability distribution and ϵ is the error parameter.
Valiant proves that the class of Boolean conjunctions and disjunctions are evolvable over uniform distribution, while Boolean parity functions are not.
The extra conditions that differentiate evolvability from the simple process of learning are as follows:

1. At each step of learning a choice of a hypothesis function from a polynomial size set of hypothesis.

2. Tolerates at most a small decrease in performance.

3. The choice of the next hypothesis from among the candidate hypothesis is made on the basis of the aggregate performance on inputs and not differentially according to the values of the various inputs.

Having established a mathematical framework for evolution, Valiant proves that the function class of Disjunctions and Conjunctions are evolvable over uniform distribution. He also proves that parity functions are not evolvable. Learning an arbitrary parity function over n variables would require either the number of generations or the population size to be exponentially large in terms of n.
A diagrammatic chart of the genetic algorithm formulated by Valiant is given below.

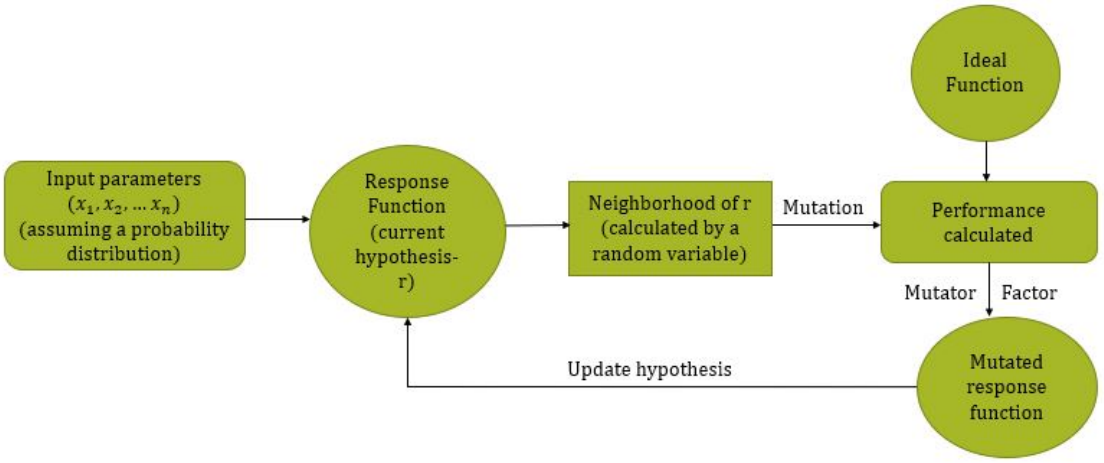


FIGURE 2.3: Valiant's algorithm for learning a function

2.3 Alon's Framework

Alon, unlike Valiant, focuses on characterization of mutations that would lead to modularity while conducting simulations. His model has a greater biological significance as it takes into account the mechanism of gene expression which directly affects the response of an organism. The model consists of an intermediate layer between the condition variables and the representation/response function. This layer takes into account the role transcription factors (TFs) play in the expression of any gene.

2.3.1 Model

Alon works in the space of square matrices which have been widely used to model gene regulation. The response function defined in Valiant's framework is analogous to the multiplication of two matrices A and B where the elements of $A = (a_{ij})_{n\times n}$ represent the regulatory strength of gene i by TF j and elements of $B = (b_{ij})_{n\times n}$ represent the effect of signal j on TF i. Here, conditions are translated in terms of signals 's' that affect the working of transcription factors which in turn affect the expression of genes. A goal matrix G is defined to be the matrix with all ideal values regarding gene expression given any signal s. The response values are in terms of gene expression vectors that are calculated as - ABs and Gs, given any input signal s. The quantity Gs is analogous to the functional value of an ideal function. The fitness measure F is

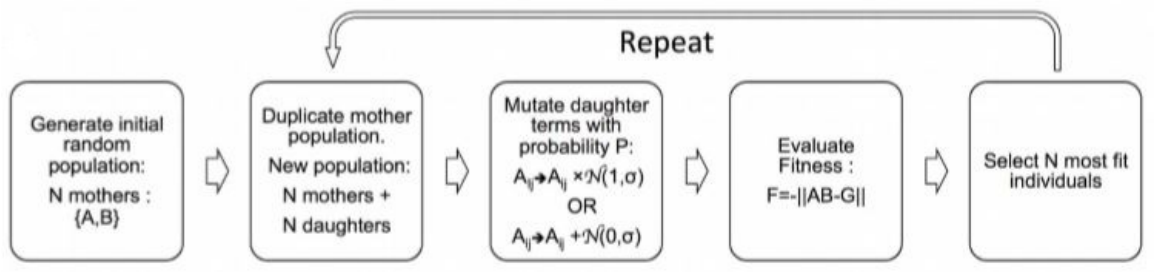


FIGURE 2.4: Evolutionary Algorithm (Figure from [3])

given by the difference between the two response vectors ABs and Gs or the difference between the response matrix and the goal matrix i.e. $F = -\|AB - G\|$ where $\|\cdot\|$ is the Euclidean norm. This measure is equivalent to the performance in the Valiant's setup.

Simulations done by Alon involve duplication, mutation followed by the selection the fittest individuals from a population of N individuals. Every iteration involves the following steps:

1. Duplication of the population
2. Mutation of the daughter population
3. Calculation of fitness
4. The N most fit individuals are selected that form the mother population for the next iteration

This iterative process (2.4) is repeated till we are unable to maximize the fitness any further or matrices A and B become block matrices which are perceived to be modular structures.

Alon focuses on two types of mutations - sum-rule mutation and product-rule mutation. Sum-rule mutation involves adding a Gaussian noise to randomly chosen elements of matrices A and B. Whereas product-rule mutation involves multiplying randomly chosen elements of the matrices with some Gaussian noise.

He proved that irrespective of the fact that the goal is modular or not product-rule mutations lead to modularity (modular structures). Whereas the sum-rule mutation doesn't lead to modularity in any case. The reason behind product-rule mutations pushing the model towards modularity is that it reduces the interaction terms and keeps small interaction terms small. It is more likely to reduce an interaction term rather than increase it due to the shape of the Gaussian. This causes the model to approach structures that have optimal fitness with minimal number of interactions (maximal number of zeroes in the matrices). When the goal is modular this in turn leads to a modular structure.

2.4 Conclusion

Assuming that the theory given by Wagner about modularity is true, we want to find out if it is possible to validate this theory in a mathematical setup like Valiant's or Alon's. Another question that is raised in the course of this thesis is that is modularity beneficial in a way for an organism? The intuitive answer would be yes because changes in a modular system don't require a costly global reorganization and small changes made to the target modules would do the job in a very cost-effective way. What remains is to prove this mathematically.

Valiant on the other hand, describes the concept of modularity as a consequence of limitations of evolvability. There are innumerable characteristics that have evolved over thousands of generations and hence should be realized as evolvable functional classes. But he suggests that according to his theory and the results proved in his paper the functional classes that are evolvable are very limited. So, we speculate that the systems that would evolve would be constrained to be modular. Each module in the system would be a functional complex that would have evolved to one of the functions from an evolvable class of functions. The different types of structural integration would give rise to structures with more complex functionalities, functions which do not belong to any class of evolvable functions. But this has yet to be proven.

In this research we formulate a mathematical model from both the models studied above and try to answer the questions posed relating to modularity, its origin and its benefits by conducting simulations using the algorithm described in [3].

Chapter 3

Simulations of Network Evolution

3.1 Modularity Metric and Factors Affecting Evolution of Modularity

The first step to answering the questions posed in the previous section is characterizing factors that play a role in the evolution of modularity. One such factor (also studied by Alon) is the type of mutation. Alon [1] studies two different types of mutation; sum-rule mutation and product rule mutation. Both mutations involve a random number being added or multiplied with randomly picked elements of the matrices.

- **Sum-Rule Mutation**: Random numbers are sampled from a Gaussian centered at the additive identity-0 with standard deviation σ. For a randomly chosen element A_{ij} and B_{ij} of matrix A and B respectively, the sum rule is as follows:

$$A_{ij} \rightarrow A_{ij} + \mathcal{N}(0, \sigma)$$

$$B_{ij} \rightarrow B_{ij} + \mathcal{N}(0, \sigma)$$

- **Product-Rule Mutation**: Random numbers are generated from a Gaussian centered at the multiplicative identity 1 with standard deviation σ. For a randomly chosen element A_{ij} and B_{ij} of matrix A and B respectively, the product rule is as follows:

$$A_{ij} \rightarrow A_{ij} \cdot \mathcal{N}(1, \sigma)$$

$$B_{ij} \rightarrow B_{ij} \cdot \mathcal{N}(1, \sigma)$$

It is important to note that the types of mutations are characterized based on the operation being performed and not the distribution used for sampling. Alon studied two different distributions in case of product-rule mutations, Gaussian and log-normal, and found that the results were

[1] [3]

qualitatively similar. For the purpose of our research we only used Gaussian distributions.
It seems intuitive to think that factors that could potentially affect the evolution of modularity would have a direct impact on the modularity metric and so it is essential to study the change in the latter while varying the former. There are many different metrics defined for modular structures but the metric used in [3] has been defined exclusively for diagonal matrices. In the setup defined in [3], diagonal and block-diagonal matrices are analogous to modular structures. Modularity (M) for any **diagonal matrix** is defined as

$$M = 1 - \frac{\langle |\ n\ | \rangle}{\langle |\ d\ | \rangle} \tag{3.1}$$

where $\langle |\ n\ | \rangle$ and $\langle |\ d\ | \rangle$ are the mean absolute value of the non-diagonal and diagonal terms respectively. The rows and columns are permuted to maximize the modularity (M). Note that $M \in [0, 1]$ where $M = 1$ for a diagonal matrix and $M = 0$ for a matrix with all equal entries. Modularity is calculated for each organism in the population and then the average is taken over the entire population. The quantity obtained is used to conclude whether the population is modular or not.
The maximum number of zeroes present in matrices A and B together is calculated using the formula $D^2 - D + k$ where D is the dimension of a full-rank goal matrix G and k is the number of zeroes present in G. Note that there can be many different ways in which these zeros are distributed among elements of A and B. In our simulations, we found that in case of product-rule mutations, these zeroes are equally distributed between the two matrices.
We study the dependence of the factors on different types of modularity metrics later on in this chapter.

3.2 Simulations

3.2.1 Initialization

The matrices A and B were randomly initialized using a uniform distribution with support $[0, 1]$. The goal matrix G was initialized to a diagonal matrix.

$$G = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}$$

The maximum number of generations were fixed, 800 in number and the population size (N) set to 500. The variance for sum-rule and product-rule was set to 0.05 and 0.27 respectively. The mutation rate is dependent on the dimensions (D) of the goal matrix and was set to $\frac{0.05}{D^2}$.
The selection mode was set to 'Tournament'. In this mode N number of sets are uniformly sampled (with repetitions) from the population containing s number of individuals. The fittest individual in each set moves on to the next generation while the rest die out. The selection intensity is directly proportional to s i.e., the number of individuals sampled in each set, higher

the value of s, greater the selection intensity. The value of s was fixed to 4.
Since attaining perfect fitness ($F = 0$) would take infinite time, a stopping criterion is defined using ε. The algorithm stops once the fitness values becomes less than or equal to ε. In our simulations, $\varepsilon = 0.01$.

3.2.2 Results

A comparative study of sum-rule and product-rule mutations required all other factors that could affect evolution of the population in any way to be fixed. Our initialization of the genetic algorithm was similar to the one studied in [3].

- **Convergence**: The algorithm converged for both sum-rule and product-rule mutations well under 800 generations (figure 3.1). The population reaches a fitness closer to the optimal in case of product-rule mutations than the fitness reached in case of sum-rule mutations. However, the number of generations needed to converge is comparable for both types of mutations. This suggests that for the given set of initialization, the rate of convergence is independent of the type of mutation.

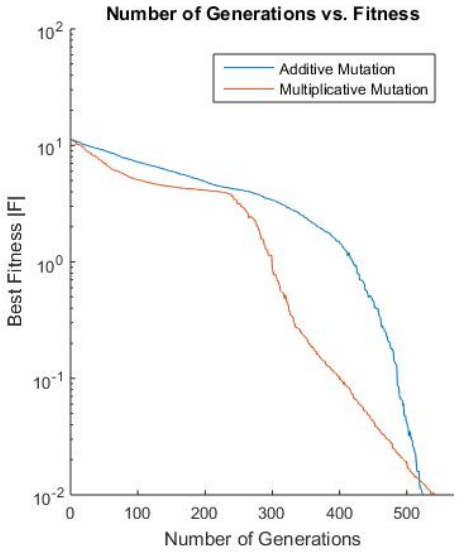


FIGURE 3.1: Comparison of fitness of populations evolving through sum-rule and product-rule mutations

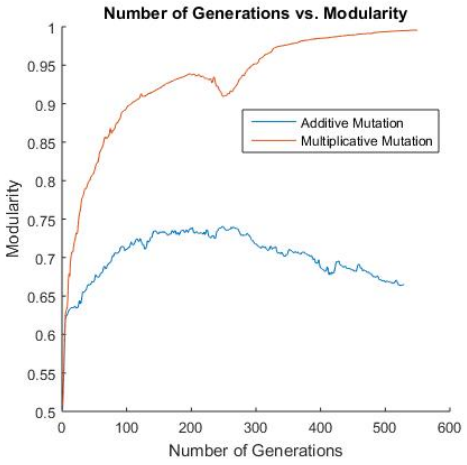


FIGURE 3.2: Comparison of modularity of populations evolving through sum-rule and product-rule mutations

- **Modularity**: Figure 3.2 is a plot of modularity plotted for each generation. For every generation, modularity is calculated for every individual and then averaged over the entire population. Clearly, simulation employing product-rule mutation to evolve the population converged to a modular solution whereas the simulation using sum-rule mutation did not. This is in accordance with the results documented in [3]. Product-rule mutations lead to modularity but sum-rule mutations do not.

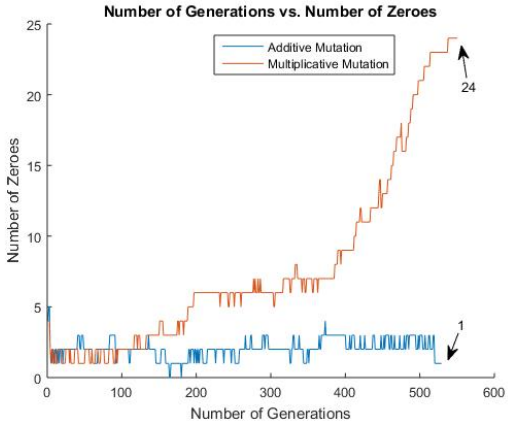


FIGURE 3.3: Comparison of number of zeros in a population evolving through sum-rule and product-rule mutations

- **Number of Zeros**: Since the goal is a full rank diagonal matrix of dimension 4, the **maximum** number of zeros that the evolved population of matrices A and B (together) can have is 24 by the formula given in one of the previous sections. Figure 3.3 is a plot of average number of zeros in the entire population at every generation. Clearly, product-rule mutations lead to a population for which the average number of zeros is equal to the maximum number of zeros possible. This suggests that the evolved population is modular. Although we cannot conclude anything about the distribution of these zero terms from this plot.

  Sum-rule mutations on the other hand lead to a population having only one zero term on average. Clearly the evolved population is not modular. It is important to note that the decomposition of the goal matrix G into two matrices A and B is a degenerate problem and can have multiple answers. Product rule-mutations lead the algorithm towards the modular solution present in the solution space whereas sum-rule mutations do not.

  To study the distribution of these terms the following plots were generated.

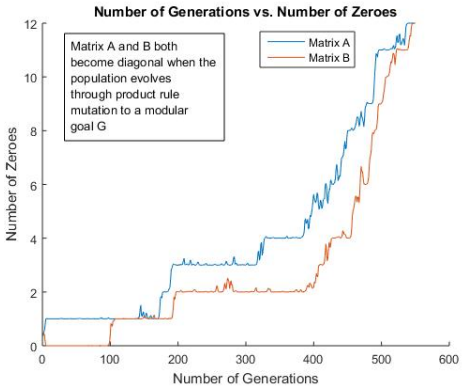


FIGURE 3.4: Number of zeros in matrix population A and B evolving through product-rule mutation

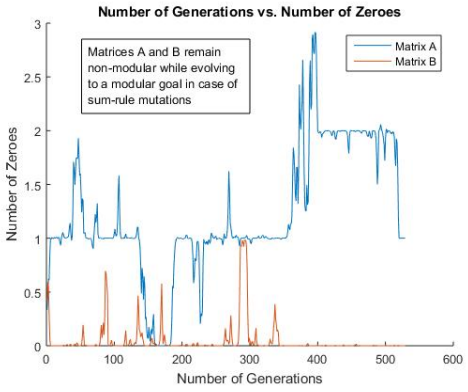


FIGURE 3.5: Number of zeros in matrix population A and B evolving through sum-rule mutation

Figure 3.4 is a plot of average number of zeros in matrix A and B populations at every generation when product-rule mutations were used. Similar plot was generated for when

sum-rule mutations were used to evolve the population (figure 3.5).
From figure 3.4, it is clear that as the number of generations progress the number of zeros in the population also increase. In fact, these terms are equally distributed between matrix A and matrix B. With 12 zeros each, A and B are diagonal matrices and hence are modular. This further validated the theory proposed in [3] that product-rule mutations lead to modular solutions. The steady increase in the number of zeros is not a coincidence and can be explained mathematically. Product-rule mutations keep small interactions terms small and hence once a near-zero interaction term is attained the next mutation will tend to keep it small. This can be explained in terms of the Gaussian distribution from which product rule mutations are sampled.

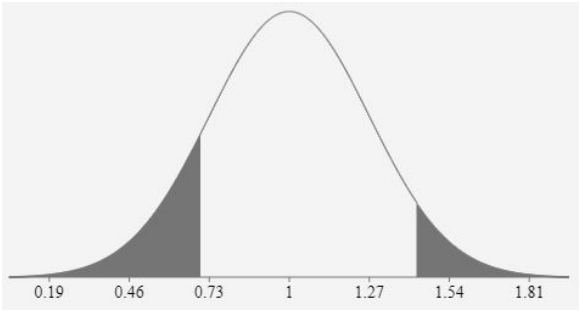


FIGURE 3.6: Probability density function of $\mathcal{N}(1, 0.27)$

Figure 3.6 shows a probability density function of a Gaussian distribution with mean 1 and standard deviation 0.27, as used in our simulations for sampling multiplying factors in product-rule mutations. The shaded areas give a certain probability associated with sampling a number from the shaded part of the number line. Let us consider an example, say an interaction term has value 3 and it gets mutated by multiplication with a random number 0.7 or less (whose sampling probability is given by the left shaded region and is equal to 0.1333). The resultant mutated interaction term would be 2.1 or less. Now to undo this mutation a random number greater than or equal to 1.43 needs to be sampled from the Gaussian. However, the probability of that happening is much lower (equal to 0.0556) and is given by the right shaded area. Also, strictly zero terms become fixed points in case of product-rule mutations. Hence, in probabilistic terms it is far more likely that a small number remains small than increases after being mutated and so the matrix tends to become sparse.

This, however, is not true for sum-rule mutations, as can be seen from figure 3.5. Sum-rule mutations, unlike product-rule mutations, show a constant drift rate whatever may be the value of the elements because the Gaussian is centered at 0. This is explained by the high fluctuation seen in the number of zeros in the population in the plot. The evolved population is neither modular nor sparse. Matrix B population has no zero terms on average whereas matrix A population has one zero term on average. The results obtained are in confirmation of the theory that sum-rule mutations do not lead to modular solutions even if the goal is modular.

3.2.3 Dependence on Modularity Metric

Modularity is calculated by the equation 3.1. There are different ways of calculating the mean absolute value of the diagonal terms i.e. $< |d| >$. We proposed two different variants (version 2 and 3) of this algorithm for calculating modularity.

1. **Version 1**: is the algorithm used in [3] (labeled as 'v.1' in Figure 3.7). The algorithm used in involves permuting the rows of matrix A and columns of matrix B such that $< |d| >$ is maximized. At each generation, D (dimension of the matrices) largest terms of the product of the matrix AB were taken to be diagonal and the rest $D^2 - D$ terms were taken to be non-diagonal.

2. **Version 2**: involved calculating the D largest terms in both matrices separately at each generation to calculate the value of $< |d| >$ and the average was taken over both matrices (labeled as 'v.2')

3. **Version 3**: of the algorithm involved calculating the elements involved in the D largest terms in the product AB. The rest of the terms were taken to be non-diagonal and their average was taken over the elements and the two matrices (labeled as 'v.3').

Simulations were repeated for the initial settings described in section 3.2.1 using the three different algorithms for calculating modularity explained above. The following plot was generated– It is clear from the figure that the result corresponding to modularity discussed in section 3.2.2

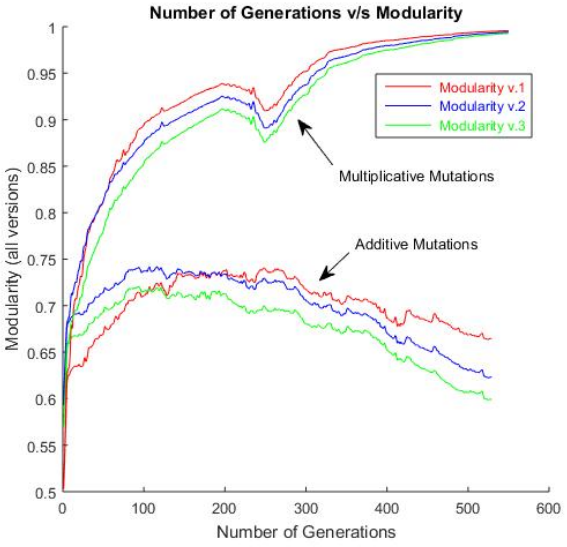


FIGURE 3.7: Comparison of all three versions of the algorithm for calculating modularity

is independent of the algorithm used for calculation of modularity. No matter what algorithm is used, product-rule mutations lead to modular solution while sum-rule mutations don't. Hence, Alon's theory is robust to different versions of modularity calculation.

3.3 Conclusion

The results obtained from our simulations are in accordance with theories proposed in [3]. The type of mutation used for evolving a population directly affects the modularity of the solution. It is important to note that only diagonal matrices were studied as the modularity defined by Equation 3.1 is applicable only for diagonal matrices. Even though the results were robust to the version of algorithm used for calculating modularity, small deviations were still registered. So, the metric used for modularity was changed to a more general notion: Newman-Girvan modularity [4]. **All further calculations of modularity metric in this thesis are done using the Newman-Girvan formula (as described in [4]), explained in Section 4.1.2, unless specified otherwise.**

Chapter 4

Amalgamation of Simulation and Theoretical Frameworks for Evolvability

4.1 Introduction

Valiant [5] proved that the class of conjunctions and disjunctions are evolvable over uniform distribution whereas parity functions are not. He explained this limitation or inability to evolve parity functions with modularity. He claimed that modularity is a consequence of limitations of evolvability, i.e. to evolve what is believed to be not evolvable, modular structures are needed. This theory hasn't been proved or disproved yet and doing so would require simulating evolution. We studied the similarities and correspondences between the two frameworks and extended the genetic algorithm described in [3] to Valiant's framework.

4.1.1 Correspondences

Frameworks described in [1] and [3] are related in many aspects, documented in the table below (Table 4.1). The goal matrix G and the ideal function f are analogous to each other as both of them are targets that the algorithms eventually should converge to. The objective functions - performance ($Perf_f(r, D_n)$) and fitness (F), of the optimization algorithms are although defined differently have the same meaning. Both measures quantify the closeness of an organism to the ideal situation/response.

Given these correspondences it becomes easier to transform the space of matrices (Alon's setup) to the space of Boolean functions (Valiant's setup). Although there is a subtle representation difference between a pair: external condition x and external signal s. The former is expressed explicitly in the setup and used to define the ideal and hypothesis Boolean functions. Whereas the latter is more implicit in it's definition as the norm is taken over the entire space of external

Valiant's Framework	Alon's Framework
x	s
$f(x)$	Gs
$r(x)$	ABs
$Perf_f(r, D_n)$	$F = -\|\|AB - G\|\|$
M_1	mutation rules
mutator factor	selection rule

TABLE 4.1: Correspondences of variables defined in Valiant's and Alon's mathematical setup

signals. Despite these subtle differences substituting s with x in the matrix setup does make sense as both of them represent external environment but at different levels of specificity; x is more general than s.
The purpose of this transformation is to extend the framework given in [3] to the framework given in [5] so that the genetic algorithm can be used to simulate the evolution of functions.

4.1.2 Newman-Girvan Modularity

Newman-Girvan modularity (Q) metric is used for characterizing community structure of networks wherein, a module or a community can be defined as a sub-graph or sub-network with substantially higher density of links than the entire network as a whole. The formula given below is for undirected and unweighted networks.

$$Q = \frac{1}{2M} \sum_{i,j=1}^{N} \left\{ \left(A_{ij} - \frac{k_i k_j}{2M} \right) \delta(C_i, C_j) \right\} \tag{4.1}$$

Here, M is the total number of links and N is the total number of nodes. The degree of i^{th} node is given by k_i. C_i is the community i^{th} node belongs to and A_{ij} is the adjacency matrix corresponding to the network. $\delta(\cdot, \cdot)$ represents the Kronecker delta function. The number of communities is fixed beforehand and calculations are repeated for different number of communities. The Kronecker delta function restricts the summation over node pairs (i, j) that belong to the same community. The algorithm involves optimizing the quantity $\left(A_{ij} - \frac{k_i k_j}{2M} \right)$ such that value of Q is the highest. This term penalizes those pairs which belong to the same community but are not connected. Due to the normalizing term $2M$, $Q \in [0, 1]$. High values of Q indicate presence of dense modules or communities but do not indicate the number of such modules.

4.2 Function Definitions

Transformation of the space of matrices to the space of Boolean functions needs to be such that it retains the matrix multiplication model, the significance of which is explained in later sections of this chapter. The most intuitive way to do so is by replacing the external signal s by external condition x in the matrix multiplication format. Note that this substitution is only

possible when the variable x is defined as a vector whose dimensions make the multiplication well defined. Then each element of the vector becomes and external condition and $\mathbf{x}$ is now a vector of external conditions. The value obtained from multiplication is still not binary. So, sign function is applied to constrain the real value to Boolean. Hence, the response function is now defined as follows:

$$r(\mathbf{x}) = sign(AB\mathbf{x}) \tag{4.2}$$

Similarly, the ideal function f can be defined in terms of the goal matrix G.

$$f(\mathbf{x}) = sign(G\mathbf{x}) \tag{4.3}$$

It is important to note that the entries in matrices A and B are still real valued (support [0,1]) and the mutation scheme is the same as described in [3]. Also, it is not necessary to define an ideal function as in 4.3. A Boolean function can also be taken as an ideal function.

The size of the matrices can be manipulated to allow for two different types of ideal functions; single target and multi-target. We conducted simulations for both the types of functions and compared the results obtained for sum and product mutations.

4.2.1 Single Target Functions

A single target function is a Boolean function of the input vector $\mathbf{x}$. We studied the case of disjunctions as single target functions. The variable $(\mathbf{x})$ was defined as $(n \times 1)$ Boolean vector. The matrix A was defined as a row vector of size $(1 \times n)$ and matrix B was taken to be a square matrix of order n. Note that for n number of condition variables, the different values vector x can take are 2^n in number i.e. X_n, the set of all Boolean vectors of size n has cardinality 2^n. The target function was defined as disjunction of n variables. We conducted simulations for $n = 3$.

$$f(\mathbf{x}) = x_1 \vee x_2 \vee x_3 \tag{4.4}$$

where,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The definition of the hypothesis functions (r) remained the same as in equation 4.2. Instead of the fitness, performance was calculated, assuming a uniform distribution over the condition variables. The performance was normalized so that it's value doesn't exceed 1. Hence equation 2.1 was modified to:

$$Perf_f(r) = \frac{1}{2^n} \sum_{\mathbf{x} \in X_n} f(\mathbf{x}) r(\mathbf{x}) \tag{4.5}$$

The evolutionary algorithm described in Alon's paper was used to evolve the hypothesis function as close to the single target ideal function as possible. It is important to understand that the algorithm still worked in the space of matrices and that the calculations of values of hypothesis

functions and performance were done independently using the population attained at every iteration. Hence, at every iteration we got a population of matrices and a population of hypothesis functions corresponding to it.

All parameter settings were retained from the previous simulations except the range of values of elements of matrices A and B. Previously, while working with matrices, the range was kept to be $[0, 1]$. But now with Boolean functions, 'sign' partitions the space of values taken by $AB\mathbf{x}$ into negative and positive values. The range was changed to $[-1, 1]$ from $[0, 1]$ so that there is no bias towards positive values.

The simulations were conducted for different values of the mutation sizes (the variance of the Gaussian from which mutation values are sampled) ranging from 0.05 to 0.5 for both, multiplicative and additive mutations. What is interesting to note is that in none of the simulations the algorithm converged i.e. the population of hypothesis functions was unable to evolve to or learn the ideal function. This is precisely due to the structure of the condition variables ($\mathbf{x}$). The definition of $\mathbf{x}$ doesn't include a dummy variable and as explained in the section below it is vital for convergence of the algorithm.

4.2.1.1 Importance of a Dummy Variable

We use an example to demonstrate the importance of a dummy variable. For simplicity, consider X_2, the space of all (2×1) Boolean vectors. Let the ideal function be the conjunction of x_1 and x_2 i.e. $f(\mathbf{x}) = x_1 \wedge x_2$. The hypothesis function is defined using 4.2. The matrix multiplication can be represented in terms of a linear function and so, the hypothesis function becomes

$$r(\mathbf{x}) = sign(AB\mathbf{x}) = sign(w_1x_1 + w_2x_2)$$

where w_1 and w_2 are the coefficients of x_1 and x_2 respectively. Note that the values of these coefficients directly influence the output of the function. Without loss of generality, four cases are possible (of the values coefficients can take) and are listed below.

1. $w_1 > w_2 > 0$
2. $0 > w_1 > w_2$
3. $w_1 > 0 > w_2$ with $|w_1| > |w_2|$
4. $w_1 > 0 > w_2$ with $|w_2| > |w_1|$

We studied only the first two cases. Similar results can be extended to the other two cases as well.

The highlighted rows of the tables above are the instances when the hypothesis function doesn't match with the ideal function. This shows that given any value of the coefficients it is not possible to learn the ideal function (in all cases) and hence the genetic algorithm will never converge irrespective of the initialization. So, to counteract the sensitivity of the functional values to the coefficient values we need to include a dummy variable $x_0 = 1$. Note that adding

x_1	x_2	$w_1x_1 + w_2x_2$	$r(\mathbf{x})$	$f(\mathbf{x})$
1	1	> 0	1	1
1	-1	> 0	1	-1
-1	1	< 0	-1	-1
-1	-1	< 0	-1	-1

TABLE 4.2: Case 1 : $w_1 > w_2 > 0$

x_1	x_2	$w_1x_1 + w_2x_2$	$r(\mathbf{x})$	$f(\mathbf{x})$
1	1	< 0	-1	1
1	-1	< 0	-1	-1
-1	1	> 0	1	-1
-1	-1	> 0	1	-1

TABLE 4.3: Case 2 : $0 > w_1 > w_2$

the dummy variable changes the size of the matrices but not of X_n. Also, the definition of the ideal function changes only if it is defined as in 4.3. Now the condition variable for this example becomes,

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

or more generally for n condition variables,

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{4.6}$$

where $x_0 = 1$. The matrix A is now a row vector of size (1×3) and matrix B is a square matrix of order 3. In general, the condition variable $\mathbf{x}$ becomes a $((n+1) \times 1)$ vector, matrix A becomes a $(1 \times (n+1))$ row vector and matrix B becomes a square matrix of order $(n+1)$.

The dummy variable was added and the simulations were repeated. The ideal function was defined as in 4.4. The parameter settings were taken as defined in section 3.2.1. The algorithm converged for both types of mutations and the perfect performance measure $(= 1)$ was also attained by the populations. All the simulations conducted are robust as the results were averaged over 100 different random intilizations of the populations.

We also studied the effect of different sizes of mutations on convergence. The ideal function, hypothesis function and parameter settings except the mutation sizes were kept the same. The following plot was generated:

Figure 4.1 shows the average number of generations needed to converge for different mutation sizes. Due to the high fluctuations in the graphs corresponding to both mutations it is difficult to accurately predict a pattern. From this plot we cannot conclude what effect does mutation size on the rate of convergence.

4.2.2 Multi-Target Functions

A multi-target function is a vector of functions, i.e., the function outputs a vector rather than a single value for each input vector $\mathbf{x}$. In our setup, the sizes of matrices A and B can be

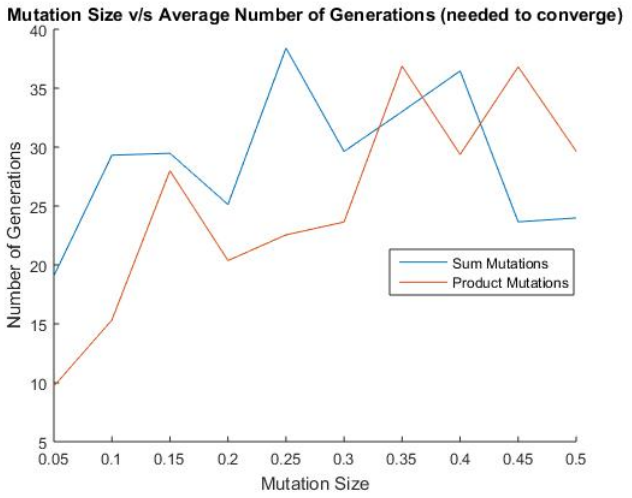


FIGURE 4.1: Average number of generations the algorithm takes to converge for every mutation size (σ)

manipulated in equation 4.2 so that the hypothesis function becomes multi-target. Multi-target ideal functions can be defined using a matrix as well (4.3), although, using a matrix is not necessary. While defining the ideal function using a goal matrix it is important to keep in mind that the orders of G and the matrix AB match otherwise the ideal function and hypothesis functions will be of different sizes and the performance will not be defined.
Ideal function and hypothesis functions were taken to be multi-target and to allow for this change, matrix A was converted to a square matrix of order $n+1$ from a row matrix of size $(1 \times (n+1))$. In general, the hypothesis function $r(x)$ and ideal function $f(x)$ for 'n' number of condition variables are given by:

$$r(\mathbf{x}) = \begin{bmatrix} r_1(\mathbf{x}) \\ r_2(\mathbf{x}) \\ r_3(\mathbf{x}) \\ \vdots \\ r_{n+1}(\mathbf{x}) \end{bmatrix} \quad \text{and} \quad f(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \\ \vdots \\ f_{n+1}(\mathbf{x}) \end{bmatrix}$$

where $r_i(\mathbf{x})$ and $f_i(\mathbf{x})$ are single target Boolean functions $\forall\, i$. The algorithm involved learning $f_i(\mathbf{x})$ through $r_i(\mathbf{x})$, $\forall\, 1 \leq i \leq n+1$. The dummy variable was retained in this setting as well because of the reasons explained in the above section. The sizes of the space X_n and matrix B remain unchanged.

Since the functional value of the ideal function and hypothesis function changed from a Boolean value to a Boolean vector, the performance (given by 2.1) was modified accordingly so that it remains in the range $[0, 1]$. So, performance for multi-target functions, in general, is defined as follows:

$$Perf_f(r) = \frac{1}{2^n(n+1)} \sum_{i=1}^{n+1} \sum_{\mathbf{x} \in X_n} f_i(\mathbf{x}) r_i(\mathbf{x}) \tag{4.7}$$

The task of evolving or learning multi-target ideal function can be reduced to evolving matrices A and B such that their product is as close to the goal matrix G as possible. This means that the underlying task is the same as it was in Alon's setup. We conducted simulations to verify if this was true or not.

4.2.2.1 Simulations

All the parameter settings were as described in Section 3.2.1. Except, the performance metric was now defined by equation 4.7. The algorithm used to calculate modularity was the same as described in [3] and not Newman-Girvan modularity. Hypothesis functions were taken to be multi-target with three input condition variables i.e., $n = 3$. The goal matrix used to define the multi-target ideal function was the same as used in simulations of Alon's genetic algorithm i.e., $G = 2I$, where I is an identity matrix of order 4. Hence, by Equation 4.3 and 4.6,

$$f(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4.8}$$

The first term in the ideal function is due to the dummy variable $x_0 = 1$. The maximum number of generations was set to 800 with 500 individuals per generation. The mean mutation set to 0 and 1 for sum-rule and product-rule mutations respectively. The selection criterion was unchanged. The range of elements of matrices A and B was taken to be $[-1, 1]$. The following plots were obtained for the same initialization of the population as in Section 3.2.1.

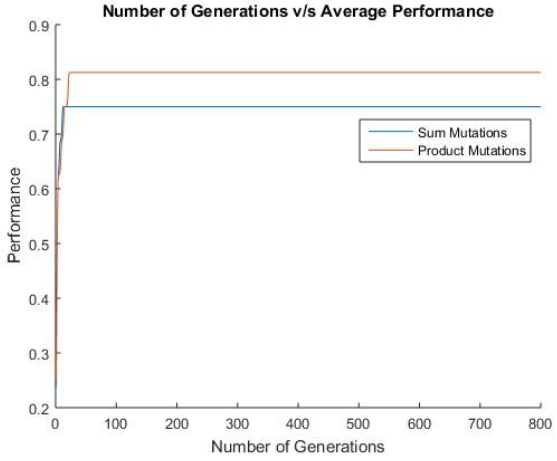


FIGURE 4.2: Average performance of the population while evolving towards the ideal function (Equation 4.8) through sum-rule and product-rule mutations.

Perfect performance was not reached instead the performance plateaued around 0.825 and 0.75 for product-rule and sum-rule mutations respectively, as seen from Figure 4.2. Clearly, the algorithm did not converge for neither of the mutations. Non-convergence of the algorithm was surprising to note as we previously discussed that the algorithm converged with the same parameter settings with fitness (F) as the objective function.

Although product mutations lead to modular solutions in the space of matrices, the same could not be concluded for this problem in the class of Boolean functions. Figure 4.3 shows the complete opposite; sum-rule mutations lead to more modular solutions than product-rule mutations.

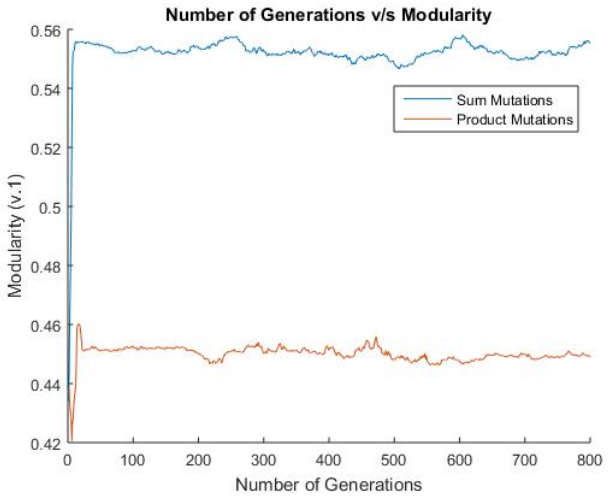


FIGURE 4.3: Modularity of the population while evolving towards the ideal function (Equation 4.8) through sum-rule and product-rule mutations.

We conducted a few experiments to check if the above case of non-convergence was an anomaly or a general trend. Simulations were repeated for different initializations of populations and qualitatively similar results were obtained. This ruled out the possibility of the case being an anomaly. Changing the parameter settings did make the algorithm converge like changing the mean mutation of product-rule mutations from 1 to 0. But this was just one hand-picked setting for which the algorithm converged. Moreover, changing the mean mutation wasn't mathematically correct as the mean should be about the identity of the operation being performed. Characterizing the problem required an in-depth analysis of every step of the genetic algorithm.

4.3 Visualization as Neural Networks

Neural Networks have long been used to carry out the task of learning of various kinds of functions. Since evolution is a form of constrained learning, the workings of the genetic algorithm and a neural network can be compared. Hence, we visualized our framework as a network and the genetic algorithm essentially became a 'learning' algorithm.

Both single target and multi-target functions were visualized as neural networks. For simplicity we studied the case corresponding to $n = 3$ (without the dummy variable). The single target function can be visualized as a neural network with one hidden layer, figure 4.4. The weight of link from X_i to z_j is b_{ji}, where $i = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$. The genetic algorithm essentially tries to evolve to those values of the weights or elements of matrices A and B such that the performance is the highest. This is analogous to how a neural network works except instead of the performance it minimizes an error function defined by the user. The optimization algorithm is also different. While we employ a genetic algorithm to carry out the task of learning, neural

networks use back propagation of error for the same and adjust the weights accordingly to minimize it (error).

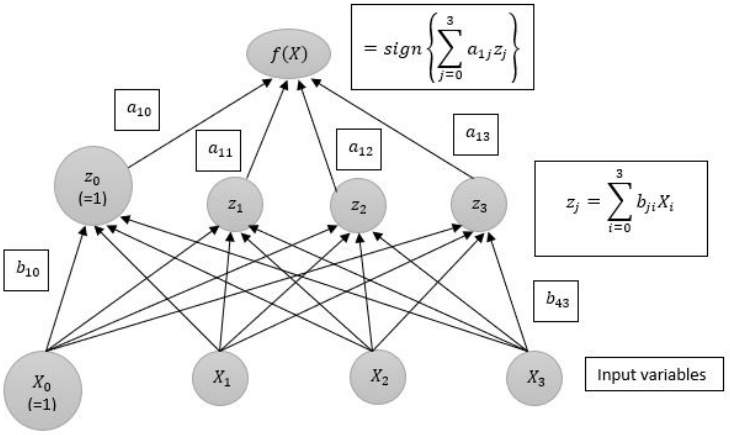


FIGURE 4.4: Neural network visualization of a single target function

The multi-target function can also be visualized as a neural network, figure 4.5. The difference in the neural network of a single target function and a multi-target function is last (output) layer. Figure 4.5 shows that the last layer has 4 nodes corresponding to each of the single target component functions. The weights for the links connecting the input layer and the hidden layer remain the same as in the case of single target functions (figure 4.4). The last layer weights, from z_i to f_j are a_{ji}. The first layer weights are the elements of matrix B and the last layer weights are elements of matrix A. In this case the number of parameters to be learnt increased by a factor of 12, or more generally $n \cdot (n+1)$ for n number of input condition variables.

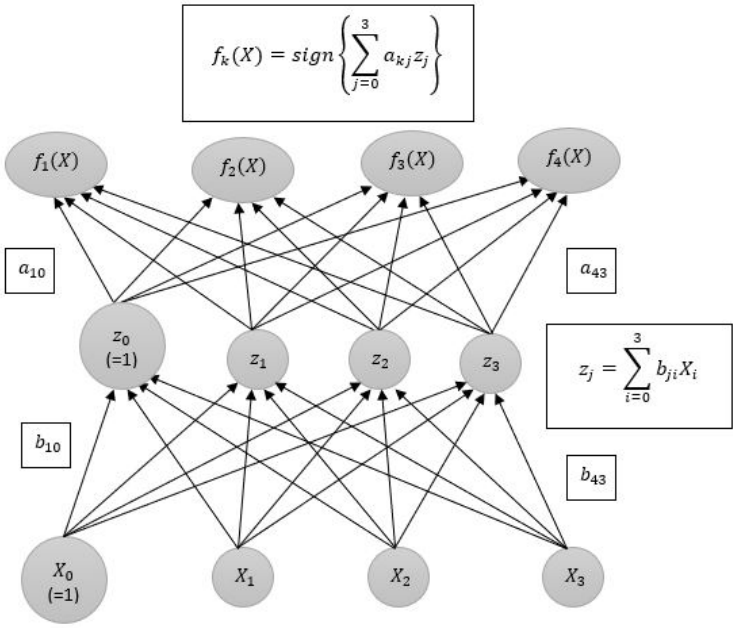


FIGURE 4.5: Neural network visualization of a multi-target function

4.3.1 Significance

In this thesis so far we have looked at linear functions only. The linearity of the space of matrices and of the sign function make it impossible to define a non-linear (hypothesis) function, for example parity functions, in this setup. But we know that the task of learning non-linear functions becomes quite simple with neural networks. The main component because of which neural networks are able to learn such functions and other techniques like SVMs cannot is the presence of hidden layers. In a similar way, matrix B is analogous to a hidden layer (Figures 4.4 and 4.5) in our setup. The presence of this intermediate stage could be manipulated to include non-linearity and enhance the evolvability of non-linear functions. This is discussed further in the next chapter.

4.4 Conclusion

Retaining the space of matrices while extending the framework to the space of Boolean functions had its advantages and disadvantages. The advantage was that despite the transformation we could still use the genetic algorithm to conduct simulations by substituting certain quantities analogous to each other and changing the sizes of some variables. The disadvantage was the redundancy added by the application of the sign function i.e. two different individuals (corresponding to a pair of matrices in the population) in the space of matrices could end up giving the same hypothesis function making the population of hypothesis functions redundant. Moreover, the small size of mutations make it difficult to mutate the hypothesis functions as changing the sign from negative to positive or vice-versa would require sampling a large number from the Gaussian. The probability of doing so is much less (Figure 3.6). So, the problem of evolving to a multi-target function is not reducible to the problem of evolving to a goal matrix as it was thought to be. We address these problems in the following chapter.

Chapter 5

Modifying the Genetic Algorithm to Study Non-Linearity

The changes made in the genetic algorithm, as described in the previous chapter, were not enough to correctly simulate evolution of functions. The redundancy in the population added due to the application of the sign function needed to be removed because it increased the time complexity of the algorithm and reduced the rate of convergence. In this chapter we study further modifications of our framework and the genetic algorithm. We also study the significance of having an intermediate layer (Section 4.3.1) by comparing one-layer and two-layer models for different Boolean functions. We focus on multi-target Boolean functions only.

5.1 Remodelling the Mutation Space and Mutation Algorithm

The current mutation space is the space of matrices with support $[-1, 1]$. The mutation algorithm involves sampling a random number from a Gaussian of a predefined mean and standard deviation. This number is then added to or multiplied with a randomly picked element of a matrix which is to be mutated. The size of the mutation is small and the application of the sign function makes it difficult to translate this change in the matrices to the level of the function. This could have been achieved by simply increasing the mutation size but it would have made the process of mutation less biologically relevant as in nature mutations occur as 'small perturbations' (as defined by Darwin). Hence increasing the mutation size beyond a certain point would not be right for the purpose of simulating evolution. To solve this problem we changed the mutation space and redefined the mutation algorithm.

The modified mutation space was a subset of the space of square matrices. The matrices A and B were now constrained to be Boolean i.e. $a_{ij}, b_{ij} \in \{0, 1\}, \forall\, i, j$. The magnitudes of these elements are no longer biologically significant as they were, before the mutation space was changed (explained in section 2.3.1). The elements capture only the interactions and not their

strengths. For example, if $b_{ij} = 1$ then TF j is activated or phosphorylated when the organism receives signal i and the TF is unaffected when $b_{ij} = 0$. Similar biological analogies can be drawn for when $a_{ij} = 0$ and 1. Further changes to the solution space were made while studying multi-target conjunctions and disjunctions (discussed later in the chapter). The space of vectors (X_n) and the function definitions remain unchanged.

The mutation algorithm was also redefined keeping the modification of the mutation space in mind. Now that the entries or elements of the matrices were either 0 or 1 the logical way of mutating them was simply flipping the values; $0 \to 1$ and $1 \to 0$. This ensured that the mutated matrices remained in the space of Boolean matrices. All other details of the mutation algorithm remained the same. The elements (of matrices) were picked randomly and mutated. Note that in this algorithm there is no use of a Gaussian. The nature of the mutation is such that it cannot be classified as product-rule or sum-rule mutation. Also, **Newman-Girvan** [**?**] metric (Equation 4.1) was used to calculate modularity.

5.1.1 Mutation Schemes Analogous to Product-Rule and Sum-Rule Mutations

Since the operation being performed is simply flipping values, the modified mutation scheme cannot be equated to sum-rule or product rule mutations. However, small modifications can make it analogous to the latter.

5.1.1.1 Multiplicative Mutation

Consider the example of a Gaussian given in Section 3.2.2. We know that product-rule mutations keep interaction terms small and have a sound mathematical explanation for the same. As demonstrated in Figure 3.6, the probability of sampling a number from the ends of the Gaussian is much lesser than sampling from somewhere closer to the mean. This makes it harder or rather less likely to reverse a mutation or increase an interaction term in magnitude. Analogously, in a Boolean setup, 'small' terms or near-zero terms can be equated to 0 and larger terms to 1. So, product-rule mutation in the Boolean setup could be imagined to have a bias which would make it harder to flip $0 \to 1$ than $1 \to 0$.

Hence we assigned a probability to every possibility. For example, assume that the mutation rate is r and the fractions of mutations that are 0 to 1 is p where $p < 1 - p$. Then there are four conditional probabilities:

- Element picked $a_{ij} = 0$
    1. Probability of a_{ij} being mutated

$$P(a_{ij} \to 1) = P_M(1|0) = r \cdot p$$

2. Probability of a_{ij} **not** being mutated

$$P(a_{ij} \to 0) = P_M(0|0) = 1 - r \cdot p$$

- Element picked $a_{ij} = 1$

1. Probability of a_{ij} being mutated

$$P(a_{ij} \to 0) = P_M(0|1) = r \cdot (1 - p)$$

2. Probability of a_{ij} **not** being mutated

$$P(a_{ij} \to 1) = P_M(1|1) = 1 - r \cdot (1 - p)$$

The conditional probabilities sum up to 1. The mutation scheme described above can be thought to be analogous to product-rule mutation in the algorithm given in [3].

5.1.1.2 Additive Mutation

Unlike product-rule mutations, sum-rule mutations don't push interaction terms in any one direction. As explained in 3.2.2, it is equally likely to decrease or increase an interaction term as the Gaussian from which random numbers are sampled is centered at mean 0. In probabilistic terms, the probability of sampling a number greater than 0 is equal to the probability of sampling a number less than 0. This is due to the symmetrical nature of the Gaussian. Analogously, in Boolean space, it would be equally likely to flip $0 \to 1$ and $1 \to 0$. This too can be modelled using conditional probabilities:

- Element picked $a_{ij} = 0$

1. Probability of a_{ij} being mutated

$$P(a_{ij} \to 1) = P_A(1|0) = \frac{r}{2} \cdot 1$$

2. Probability of a_{ij} **not** being mutated

$$P(a_{ij} \to 0) = P_A(0|0) = 1 - \frac{r}{2}$$

- Element picked $a_{ij} = 1$

1. Probability of a_{ij} being mutated

$$P(a_{ij} \to 0) = P_A(0|1) = \frac{r}{2}$$

2. Probability of a_{ij} **not** being mutated

$$P(a_{ij} \to 1) = P_A(1|1) = 1 - \frac{r}{2}$$

Note that additive mutation is a particular case of the model defined in the previous section with $p = \frac{1}{2}$ and $P(0|1) = P(1|0)$. All conditional probabilities sum to 1. The above mutation scheme is analogous to sum-rule mutations described in [3].

While comparing multiplicative and additive mutations, note that the following equation is satisfied:

$$P_M(1|0) + P_M(0|1) = P_A(1|0) + P_A(0|1) = r \tag{5.1}$$

where $P_M(\cdot)$ represents probability for multiplicative mutation while $P_A(\cdot)$ represents probability for additive mutation. This is to ensure that the net probability of mutating an element remain the same in both cases (under the assumption that 0s and 1s are equally likely to begin with).

The mutation rate r and parameter p were set before running simulations.

5.2 Evolving Multi-Target Monotonic Conjunctions and Disjunctions

Multi-target conjunctions (disjunctions) are multi-target functions in which every element is a single target conjunctive (disjunctive) function. A monotonic conjunction or disjunction doesn't have any negated literals. Evolving these functions involves learning every single target function. In order to do so the genetic algorithm traverses the fitness landscape or the solution space to converge to the right function. The solution space or the space of hypothesis functions, in this case, is the set of all multi-target (monotonic) conjunctions and disjunctions (refer to section 2.2.3). A few conditions were added on matrices A and B to restrict the Boolean functions defined by equation 4.2 to the space of conjunctions and disjunctions.

5.2.1 Solution Space

It is important to note that monotonic conjunctions and disjunctions are linear and hence can be expressed as a polynomial in condition variables (by Weierstrass Approximation Theorem). As discussed in section 4.2.1.1, these polynomials will have a bias term or a zero degree term. Consider the example of a conjunction $f(x_1, x_2) = x_1 \wedge x_2$. Then the polynomial p defined as $p(x_1, x_2) = x_1 + x_2 - 1$ and the function f are identical. The bias term (w_0) is equal to -1. Question remains how to calculate the bias term of the polynomial? There are different conditions for calculating the bias term for conjunctions and disjunctions. For an arbitrary function f with n input variables, the bias term is calculated as follows:

- If f is a **conjunction** of $0 < m \leq n$ variables i.e. $f(x_1, x_2, \ldots, x_n) = x_1 \wedge x_2 \wedge \ldots \wedge x_m$ then,

$$w_0 = 1 - m \tag{5.2}$$

- If f is a **disjunction** of $0 < m \leq n$ variables i.e. $f(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_m$ then,

$$w_0 = m - 1 \tag{5.3}$$

These polynomials can expressed in the form a matrix as well. For example,

$$f(x_1, x_2, x_3) = \begin{bmatrix} x_1 \wedge x_2 \\ x_1 \wedge x_2 \wedge x_3 \end{bmatrix}$$

can be expressed as a polynomial p where

$$p(x_1, x_2, x_3) = \begin{bmatrix} p_1(x_1, x_2, x_3) \\ p_2(x_1, x_2, x_3) \end{bmatrix} = \begin{bmatrix} x_1 + x_2 - 1 \\ x_1 + x_2 + x_3 - 2 \end{bmatrix}$$

The matrix representation of the above polynomial is as follows

$$p(x_1, x_2, x_3) = \begin{bmatrix} 1 & 1 & 0 & -1 \\ 1 & 1 & 1 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_0 \end{bmatrix} \tag{5.4}$$

The matrix representation of any multi-target function involves only one matrix whereas our model have two; matrix A and B. Also, for learning the above function f, matrices A and B will have to evolve such that their product is equal to the matrix in the equation 5.4. The only way to ensure that hypothesis functions (calculated by 4.2) do not fall outside the solution space is by restricting both matrices such that their corresponding polynomials represent one type of function (conjunction or disjunction). The logic behind this is that a conjunction/disjunction of conjunctions/disjunctions is also a conjunction/disjunction.

Generally, for learning a multi-target function f of size $m \times 1$ with n input variables, the sizes of matrix A and B used to calculate the hypothesis functions are $m \times (n+1)$ and $(n+1) \times (n+1)$. The $(n+1)^{\text{th}}$ column of matrix A and B correspond to bias terms. Also the $(n+1)^{th}$ row of matrix B is fixed (so that the dummy variable is translated even after multiplication of B and vector x),

$$b_{ij} = 0, \forall\ 1 \leq j \leq n \ \text{ and } \ i = n+1$$

$$b_{ij} = 1 \ \text{ for } \ i = j = n+1$$

Hence,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1(n+1)} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2(n+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{m(n+1)} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1(n+1)} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2(n+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ b_{n1} & b_{n2} & b_{n3} & \dots & b_{n(n+1)} \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

where,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ x_0 \end{bmatrix}$$

The position of the dummy variable x_0 was changed from the first to the last element of the vector $\mathbf{x}$ (see Equation 4.6). Note that, as discussed in the above section, both matrices are restricted to be Boolean and the bias terms are given by (calculated using Equations 5.3 and 5.2)

- If the ideal function is a multi-target monotonic disjunction

$$a_{i(n+1)} = \sum_{j=1}^{n} a_{ij} - 1 \ , \forall \ 1 \leq i \leq m \quad \text{and} \quad b_{i(n+1)} = \sum_{j=1}^{n} b_{ij} - 1 \ , \forall \ 1 \leq i \leq n$$

- If the ideal function is a multi-target monotonic conjunction

$$a_{i(n+1)} = 1 - \sum_{j=1}^{n} a_{ij} \ , \forall \ 1 \leq i \leq m \quad \text{and} \quad b_{i(n+1)} = 1 - \sum_{j=1}^{n} b_{ij} \ , \forall \ 1 \leq i \leq n$$

5.2.2 Simulations

Simulations of the modified genetic algorithm were run for different multi-target monotonic conjunctions and disjunctions. With the change in the mutation algorithm and space the parameters like mean mutation size and standard deviation became redundant. The mutation rate was set to $\frac{0.09}{n^2}$ and the maximum number of generations was taken to be 800 with 500 individuals each. The selection method was 'Tournament' with $s = 4$. The ratio of flipping $0 \rightarrow 1$ to $1 \rightarrow 0$ i.e. p (as defined in Section 5.1.1) was set to 0.3. A comparison between multiplicative and additive mutations was made and the condition (5.1) was satisfied. All simulations were robust to initialization of populations. For each ideal function 100 simulations were run with different initial population. A comparison between additive and multiplicative mutations was made.

Figure 5.1 corresponds to simulations learning multi-target monotonic conjunctions. The expression of multi-target ideal functions learnt for n inputs:

$$f(\mathbf{x}) = \begin{bmatrix} x_1 \wedge x_2 \\ x_2 \wedge x_3 \\ \vdots \\ x_{n-1} \wedge x_n \\ x_n \wedge x_1 \end{bmatrix} \tag{5.5}$$

The plot shows the average number of generations taken by the algorithm to converge to an ideal function with n number of inputs for cases corresponding to both additive and multiplicative mutations. The number of inputs was increased from 2 to 5. The average was calculated over

the converging runs out of the 100 simulations conducted. For lower values of n, the convergence rate for additive and multiplicative mutations is approximately the same with the lost being 3 generations for learning $f(\mathbf{x}) = x_1 \wedge x_2$. But as the number of inputs increase, the average number of generations needed to converge increases rapidly for multiplicative mutations.

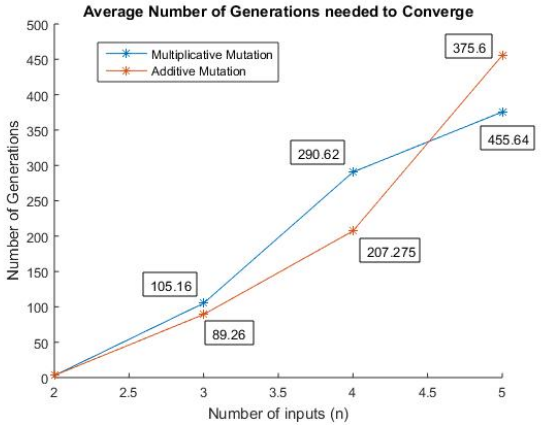


FIGURE 5.1: Average number of generations the algorithm converged in to multi-target monotonic conjunctions

We also looked at the frequency of convergence i.e. percentage of converging runs (out of the 100 simulations conducted). From Figure 5.2 it can be seen that there is a steady decline in the number of runs that converge with an increase in the number of inputs for both additive and multiplicative mutations. However, this frequency is dependent on the initialization of the population and the maximum number of generations. Increasing the number of runs per ideal function and the maximum number of generations will most likely improve the convergence frequency.

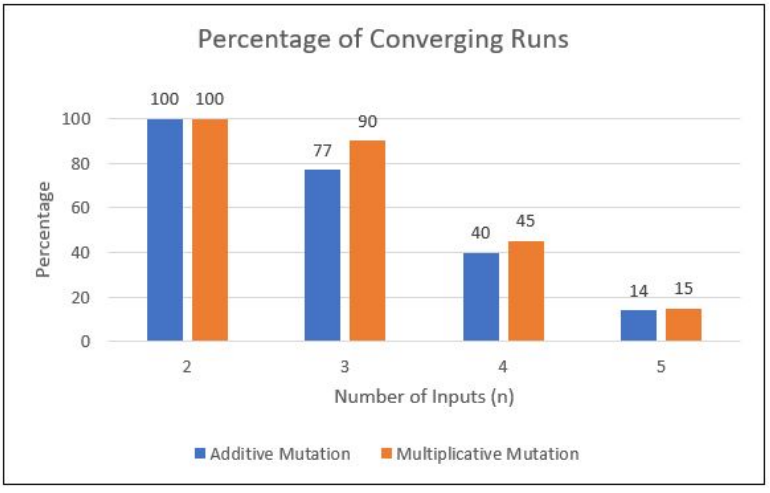


FIGURE 5.2: Percentage of converging runs in evolution of multi-target monotonic conjunctions

The results obtained for multi-target monotonic disjunctions were qualitatively and quantitatively similar to the results obtained for multi-target monotonic conjunctions. Our simulations are corroborate the theory given by Valiant in [1] that monotonic conjunctions and disjunctions are evolvable classes of functions.

5.3 Evolving Non-linear Boolean Functions

The functions studied till now have been linear and hence were easily represented in matrix form (5.4). Non-linear functions on the other hand cannot be expressed in the form of matrices due to the linearity of the space of matrices. Linear functions can be easily learnt by a neural network with one layer. Having visualized our model as an NN (Section 4.3), one might think that having a second matrix layer is redundant for learning linear functions. However, the second layer might prove to be necessary for evolving certain kinds of function classes (section 4.3.1). We know that learning a non-linear function, for example a parity function, is only possible using a neural network with a hidden layer. The property of the NN that makes it possible is the **non-linear activation function** at the hidden nodes. So, theoretically if non-linearity is added at the first layer of our model (i.e., matrix B), which is analogous to the hidden layer in a NN, it should be able to evolve non-linear functions as well. We conducted simulations to test this idea.

5.3.1 Addition of Non-Linearity

Since the space of matrices is linear, non-linear ideal functions cannot be defined by Equation 4.3. Hence, ideal functions were predefined before running simulations. Similarly, hypothesis functions defined by Equation 4.2 are linear. Since our model is analogous to a NN and matrix B to a hidden layer, adding non-linearity in hypothesis functions should be equivalent to adding a non-linear activation function. Hence, the hypothesis functions were redefined as follows

$$r(\mathbf{x}) = sign(A \cdot sign(B\mathbf{x})) \tag{5.6}$$

The second sign function is analogous to the activation function of a hidden layer and it made the hypothesis functions non-linear. This can be extended to multi-target functions as explained in Section 4.2.2. All the other details of our model remained unchanged.

The only class of non-linear Boolean functions that Valiant ([1]) talks about are parity functions. He claimed that the class of parity functions is not evolvable. So, we tested our theory for parity functions to verify Valiant's claim by conducting simulations.

5.3.2 Parity Functions

There are two types of parity functions: odd and even. An odd/even parity function is true if odd/even number of literals are true and false otherwise. XOR is an example of an odd parity function. Parity functions can be expressed as disjunctions of different conjunctive functions. For simplicity consider the example of XOR function, say $f(\mathbf{x})$, with 3 input literals – x_1, x_2 and x_3. Then,

$$f(\mathbf{x}) = f_0(\mathbf{x}) \vee f_1(\mathbf{x}) \vee f_2(\mathbf{x}) \vee f_3(\mathbf{x})$$

where

$$f_0(\mathbf{x}) = (\neg x_1) \wedge (\neg x_2) \wedge (\neg x_3)$$

$$f_1(\mathbf{x}) = x_1 \wedge (\neg x_2) \wedge (\neg x_3)$$

$$f_2(\mathbf{x}) = (\neg x_1) \wedge x_2 \wedge (\neg x_3)$$

$$f_3(\mathbf{x}) = (\neg x_1) \wedge (\neg x_2) \wedge x_3$$

So, any parity function can be expressed using AND, OR and NOT functions. The literals correspond to condition variables in our setup. These are analogous to external signals detected by an organism (Section 4.1.1). The negation of a condition variable can be thought of as the absence of that signal. Since every organism is intelligent enough to detect the presence and absence of a signal, learning the negation of one seems biologically redundant. So, technically, the size of the vector x changes from a vector of size $n+1$ to $2n+1$ to allow for negation of all condition variables, i.e.,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \\ \neg x_1 \\ \vdots \\ \neg x_n \\ x_0 \end{bmatrix} \tag{5.7}$$

The size of the set X_n remained unchanged. The dimensions of matrix B were changed from $((n+1) \times (n+1))$ to $((2n+1) \times (2n+1))$. Matrix A can be a row vector of size $2n+1$ or a matrix of size $m \times (2n+1)$ depending on the size of the ideal function. All other details of our framework and algorithm remain unchanged.

The evolvability of parity functions was studied by conducting simulations of the modified genetic algorithm.

5.3.3 Simulations

We studied single target parity functions only. It is clear from the example in the section above that in our setup any parity function can be broken down to a disjunction of various conjunctive functions. The logical or most intuitive way of progressing would be learning the conjunctive functions (f_0, f_1, f_2, f_3 in the example) first and then a disjunctive function with these functions as its input. In terms of the workings of a NN, the first layer/hidden layer, analogous to matrix B, would learn conjunctive functions while the final layer, analogous to matrix A, would be a disjunction of all the nodes of the previous layer. Hence, we conditioned matrix A and B such that they learnt only disjunctions and conjunctions respectively. This was done by manipulating the calculation of bias terms (Section 5.2.1). For matrix A,

$$a_{i(n+1)} = \sum_{j=1}^{n} a_{ij} - 1 \, , \forall \; 1 \leq i \leq m$$

Note that in our simulations $m = 1$ as we only looked at single-target functions.

For matrix B,

$$b_{i(n+1)} = 1 - \sum_{j=1}^{n} b_{ij} \ , \forall \ 1 \leq i \leq n$$

It is important to note that it is impossible to realize parity functions in terms of just one matrix (as shown in the example given in Section 5.2.1, Equation 5.4). This is because parity functions are non-linear whereas the representation as a matrix is linear in nature.

5.3.3.1 Initialization

We compared one-layer and two-layer matrix models with different ideal functions – conjunctions and odd parity functions. The maximum number of generations was set to 800 with 500 individuals per generation. The multiplicative mutation scheme was used with $\frac{0.09}{n^2}$ as the mutation rate and the fraction p of mutation $0 \to 1$ to $1 \to 0$ was taken to be 0.3, i.e. $P(1|0) = \frac{0.09}{n^2} \cdot 0.3$. Other conditional probabilities were set accordingly. The selection method remained unchanged i.e. 'Tournament' with $s = 4$. The number of input variables n was varied from 2 to 5. The modified genetic algorithm with redefined mutation and solution space was used. For every value of n and choice of ideal function, 100 runs were conducted. All simulations were robust to initialization of the population.

5.3.3.2 Results

We studied the convergence rate and convergence frequency of the final population. The following plots were generated:

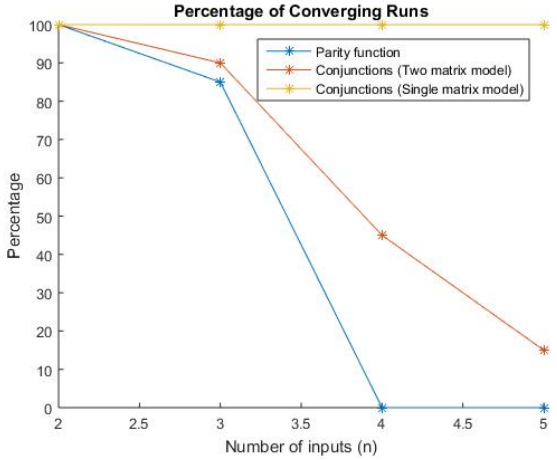


FIGURE 5.3: Percentage of Converging Runs for Parity functions and Conjunctions with n inputs evolved using **multiplicative mutation**

Figure 5.3 shows the percentage of converging runs for the 100 simulations conducted. Clearly, percentage for parity functions decline rapidly from 90% to 0% as the number of inputs are increased from 3 to 4. This means that the evolvability of parity functions decreases rapidly with increase in the number of inputs. All 100 simulations converge while evolving multi-target monotonic conjunctions, defined by Equation 5.5, using the one-matrix model. As discussed in

previous sections, adding a second layer for evolving monotonic conjunctions and disjunctions seems redundant but is absolutely necessary for evolving non-linear functions

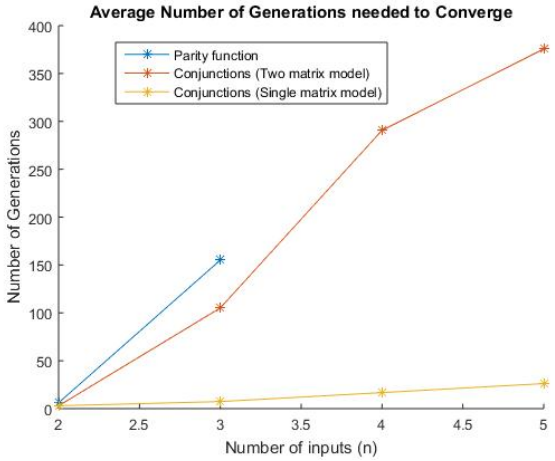


FIGURE 5.4: Average number of generations the algorithm converges in to parity functions and conjunctions with n inputs using **multiplicative mutation**

Figure 5.4 shows the average number of generations that the algorithm took to converge to parity functions and multi-target monotonic conjunctions. The plot for parity functions stops at $n = 3$ because for 4 and 5 inputs none of the 100 simulations converged. Increasing the maximum number of generations to 1000 and decreasing the number of individuals per generation from 500 to 10 did not have any impact on the convergence. All 100 runs diverged for these settings as well. There is a steady increase in the average number of generations for conjunctions evolved using a two-matrix model. However, for a single-matrix model, number of generations increase at a very slow rate, almost constant with respect to the other two.

The results for even parity functions were qualitatively similar to the results obtained for odd parity functions.

5.4 Conclusion

The above discussion together with Section 4.3.1 shows that even though a one-layer model proves more efficient for evolving monotonic conjunctions and disjunctions, having a second layer is crucial for evolving other non-linear functions which are found in many biological networks. The time complexity increases with the increase in the number of inputs (Figure 5.4). For monotonic conjunctions and disjunctions this increase seems to be linear but for parity functions it becomes exponential when n is increased beyond 3. The simulations suggest that with the present genetic algorithm it is possible to evolve parity functions only with small number of inputs. As also depicted by figure 5.3, the convergence frequency rapidly falls to 0 as the number of inputs are increased. Another surprising observation is the fall in convergence frequency for monotonic conjunctions evolved using a two-matrix model. Corresponding to the drop in convergence frequency, a sharp increase in the rate of convergence is observed which hints at the drop in the evolvability of linear functions evolved using two-matrix model. However, single matrix model

prove to be very efficient by maintaining 100% convergence and correspondingly small increase in the number of generations needed to converge even with an increase in the number of inputs. Although efficient, single matrix model can only be used to evolve linear functions. This calls into question the much speculated existence of cellular XOR gates. Even though such gates have been synthetically engineered (Bonnet *et al.* [6]) it is apparent from our results that such functions are not only difficult to evolve but also reduce the evolvability of other linear functions.

Chapter 6

Conclusion

Evolution is one the most well researched biological phenomena. It has always been viewed as a process of change of heritable characteristics of biological populations but it wasn't until recently that it came to be viewed as a process of learning. Since all organisms have the ultimate goal of survival which is greatly dependant on their responses to situations in their environment, these changes in the characteristics can be thought of as an attempt to learn the best response. Valiant was the first to introduce this idea and evolvability in terms of learnability. Evolvability is a crucial concept of evolutionary developmental biology. The mathematical setup formulated by Valiant has effectively captured this concept and uses modularity to explain its limitations. Friedlander *et al.*, on the other hand adopted a more practical approach, proved some very interesting results relating to modularity by conducting *in silico* simulations. His results show a direct relation between modularity and the nature of mutations. Our simulation results also reflected the same relation between the two. It is possible that the inability of Valiant's framework to give a proof for the relation between modularity and evolvability lies in its general setting. Since we have established many points of correspondence between Friedlander *et al.* and Valiant's framework, results obtained in Friedlander *et al.* can be thought of as a special case or conditioning of the variables in the theoretical setup given by Valiant.

We proposed a variant of the mathematical model given in Friedlander *et al.* adapting ideas from Valiant's setup. We showed that our regulatory model with multiple stages of computation between input and output is essentially equivalent to a neural network or deep learning models. Hence, the task of learning any function in this modified setup is similar to regression via neural networks. We were able to evolve monotonic conjunctions and disjunctions within a few generations, when the number of inputs was less, using the modified genetic algorithm; thus validating the theory given by Valiant. The significance of having more than one intermediate computational layer lies in evolving non-linear functions like parity functions. By introducing a non-linear layer in Friedlander *et al.* we were able to evolve parity functions with small number of inputs. However, the time complexity increased very rapidly with the increase in the number of inputs. This suggests that parity functions as a class of functions are not evolvable as claimed by Valiant but functions with smaller number of inputs can in fact be evolved thus stressing the need for an additional computational layer. Our results add weight to what Alon speculates in his

book [7]; existence of small non-linear functions in various biological networks like exclusive-OR in signal transduction networks, for example.

The next step forward would be to observe how modularity is connected to other aspects of a genetic algorithm other than the type of mutation. It would be interesting to note how varying function classes or goal matrices affect modularity. It seems intuitive that the process of learning a complex function would be greatly simplified and in turn speeded up if it is learnt as an integration of simpler functions that belong to evolvable classes of functions. These simpler functions could be expressed by modules. The aggregation of these modules would give a modular structure and this structure would collectively be responsible for the computation of the full complex function. This may explain the existence of numerous evolved characteristics despite the limitations of evolvability and the limited number of evolvable functional classes. The questions that follow this research are: 'Is modularity desirable?' and 'Is it possible that factors that enhance modularity may also have implications for evolvability?'

Bibliography

[1] Leslie G Valiant. Evolvability. *Journal of the ACM (JACM)*, 56(1):3, 2009.

[2] Gunter P Wagner and Lee Altenberg. ôcomplex adaptations and the evolution of evolvability. ö evolution. *Volume*, 50:967–976.

[3] Tamar Friedlander, Avraham E Mayo, Tsvi Tlusty, and Uri Alon. Mutation rules and the evolution of sparseness and modularity in biological systems. *PloS ONE*, 8(8):e70444, 2013.

[4] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582, 2006.

[5] Paul Valiant. Evolvability of real functions. *ACM Transactions on Computation Theory (TOCT)*, 6(3):12, 2014.

[6] Jerome Bonnet, Peter Yin, Monica E Ortiz, Pakpoom Subsoontorn, and Drew Endy. Amplifying genetic logic gates. *Science*, 340(6132):599–603, 2013.

[7] Uri Alon. *An introduction to systems biology: design principles of biological circuits*. Chapman and Hall/CRC, 2006.

[8] Vitaly Feldman and Leslie G Valiant. The learning power of evolution. In *COLT*, pages 513–514, 2008.

[9] Nadav Kashtan, Elad Noor, and Uri Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716, 2007.

[10] Vitaly Feldman. Robustness of evolvability. In *COLT*, 2009.

[11] Sumeet Agarwal. Systems approaches in understanding evolution and evolvability. *Progress in biophysics and molecular biology*, 113(3):369–374, 2013.

[12] Jeff Clune, Jean-Baptiste Mouret, and Hod Lipson. The evolutionary origins of modularity. *Proc. R. Soc. B*, 280(1755):20122863, 2013.

[13] Nadav Kashtan and Uri Alon. Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778, 2005.

[14] Nadav Kashtan, Avi E Mayo, Tomer Kalisky, and Uri Alon. An analytically solvable model for rapid evolution of modular structure. *PLoS computational biology*, 5(4):e1000355, 2009.

[15] Massimo Pigliucci. Is evolvability evolvable? *Nature Reviews Genetics*, 9(1):75, 2008.

[16] Vitaly Feldman. Evolvability from learning algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 619–628. ACM, 2008.

[17] Ingo Wegener. Theoretical aspects of evolutionary algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 64–78. Springer, 2001.

[18] Varun Kanade, Leslie G Valiant, and Jennifer Wortman Vaughan. Evolution with drifting targets. *arXiv preprint arXiv:1005.3566*, 2010.