The CRC Press International Series on Computational Intelligence

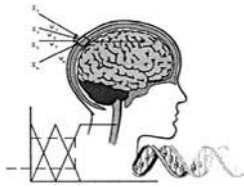
Ricardo Salem Zebulum
Marco Aurélio C. Pacheco
Marley Maria B.R. Vellasco

EVOLUTIONARY ELECTRONICS

Automatic Design of Electronic Circuits and Systems by Genetic Algorithms

EVOLUTIONARY ELECTRONICS

Automatic Design of Electronic Circuits and Systems by Genetic Algorithms



The CRC Press

International Series on Computational Intelligence

Series Editor
L.C. Jain, Ph.D., M.E., B.E. (Hons), Fellow I.E. (Australia)

L.C. Jain, R.P. Johnson, Y. Takefuji, and L.A. Zadeh
Knowledge-Based Intelligent Techniques in Industry

L.C. Jain and C.W. de Silva
Intelligent Adaptive Control: Industrial Applications in the Applied Computational Intelligence Set

L.C. Jain and N.M. Martin
Fusion of Neural Networks, Fuzzy Systems, and Genetic Algorithms: Industrial Applications

H.-N. Teodorescu, A. Kandel, and L.C. Jain
Fuzzy and Neuro-Fuzzy Systems in Medicine

C.L. Karr and L.M. Freeman
Industrial Applications of Genetic Algorithms

L.C. Jain and B. Lazzerini
Knowledge-Based Intelligent Techniques in Character Recognition

L.C. Jain and V. Vemuri
Industrial Applications of Neural Networks

H.-N. Teodorescu, A. Kandel, and L.C. Jain
Soft Computing in Human-Related Sciences

B. Lazzerini, D. Dumitrescu, L.C. Jain, and A. Dumitrescu
Evolutionary Computing and Applications

B. Lazzerini, D. Dumitrescu, and L.C. Jain
Fuzzy Sets and Their Application to Clustering and Training

L.C. Jain, U. Halici, I. Hayashi, S.B. Lee, and S. Tsutsui
Intelligent Biometric Techniques in Fingerprint and Face Recognition

Z. Chen
Computational Intelligence for Decision Support

L.C. Jain
Evolution of Engineering and Information Systems and Their Applications

H.-N. Teodorescu and A. Kandel
Dynamic Fuzzy Systems and Chaos Applications

L. Medsker and L.C. Jain
Recurrent Neural Networks: Design and Applications

L.C. Jain and A.M. Fanelli
Recent Advances in Artifical Neural Networks: Design and Applications

M. Russo and L.C. Jain
Fuzzy Learning and Applications

J. Liu and J. Wu
Multiagent Robotic Systems

M. Kennedy, R. Rovatti, and G. Setti
Chaotic Electronics in Telecommunications

H.-N. Teodorescu and L.C. Jain
Intelligent Systems and Techniques in Rehabilitation Engineering

I. Baturone, A. Barriga, C. Jimenez-Fernandez, D. Lopez, and S. Sanchez-Solano
Microelectronics Design of Fuzzy Logic-Based Systems

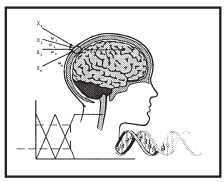
T. Nishida
Dynamic Knowledge Interaction

C.L. Karr
Practical Applications of Computational Intelligence for Adaptive Control

Ricardo Salem Zebulum, Marco Aurélio C. Pacheco, and Marley Maria B.R. Vellasco
Evolutionary Electronics: Automatic Design of Electronic Circuit and Systems by Genetic Algorithms

EVOLUTIONARY ELECTRONICS

Automatic Design of Electronic Circuits and Systems by Genetic Algorithms



Ricardo Salem Zebulum
Marco Aurélio C. Pacheco
Marley Maria B.R. Vellasco

CRC PRESS

Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Zebulum, Ricardo Salem.
Evolutionary electronics : automatic design of electronic circuits and systems by genetic algorithms / Ricardo Salem Zebulum, Marco Aurélio Pacheco, Marley Maria B.R. Vellasco.
p. cm. — (The CRC Press international series on computational intelligence)
Includes bibliographical references and index.
ISBN 0-8493-0865-8
1. Electronic circuits—Data processing. 2. Electronic circuit design—Data processing. 3. Genetic algorithms. I. Pacheco, Marco Aurélio. II. Vellasco, Marley Maria B.R. III. Title. IV. Series.

TK7867 .Z43 2001
621.3815—dc21 2001043571

No claim to original U.S. Government works
International Standard Book Number 0-8493-0865-8
Library of Congress Card Number 2001043571
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

To the previous generation, our love and acknowledgment:

Ricardo's parents
Regina
Carlos

Marco Aurélio's parents
Maria Leônia
Ageu

Marley's parents
Maria Heloisa
Marzio

PREFACE

Many books on Evolutionary Computation, and, more specifically, on Genetic Algorithms, have already been published. When we decided to write a book on Evolutionary Electronics, we were basically motivated by what appears to be a turning point in this area: a wider range of applications in synthesis of engineering and control systems, being now tackled by evolutionary computing techniques. Other stimulating reasons were: the investigation of advanced paradigms, ranging from new techniques for multi-objective optimization to more biologically plausible algorithms; the steady increase in the processors' speed, which is now making possible the sampling of larger search spaces; and, finally, the emergence of complex technologies' promises to turn evolutionary computation into a valuable tool to be employed in the design process of many systems.

Evolutionary Electronics is a research area which involves applications of Evolutionary Computation in the domain of electronics. This new field of research has shown a potential which is attracting not only researchers, but also electronic circuits manufacturers, technology developers, end-users from the aerospace community, and engineers. Their motivations are various: the need of high performance electronic circuits with more sophisticated specifications; the increasing number of requirements that must be satisfied by a design in electronics; the appearance of more complex technologies; the availability of mature reconfigurable circuits platforms; the trend towards the achievement of fast, low-power and low-area integrated circuits; the issue of fault tolerant systems for harsh environments; and the high cost of experienced electronic designers, mainly in analog electronics. If, on the one hand, one can doubt that standard methodologies will be able to cope with the increasing complexity of the area, on the other hand, evolutionary computation is seen as the most promising alternative to overcome the drawbacks of conventional design. Any tool that can handle these previous issues will be widely applied in industrial control systems, high performance communications systems, and in space exploration missions.

This is the first book dedicated to Evolutionary Electronics, and it will be certainly followed by many others. Nonetheless, the scope of this book is not restricted to Evolutionary Electronics; our purpose is also to introduce new methods for Evolutionary Computation which can be applied to the synthesis and system design in other engineering fields. These new methods differ from the already existing ones due to aspects of biological plausibility and the need for better results in real applications. Since the motivation to develop artificial evolutionary algorithms has been to reproduce, in a simplified way, natural evolution mechanisms, we go a little deeper into this issue, trying to incorporate more aspects exhibited by natural systems in our Evolutionary Algorithms. When we talk about tackling real applications, we mean that our methods are not only intended to be applied in the electronics world, but to any real design problem that can be interpreted as a search problem. Since real problems usually involve the sampling of a large number of potential solutions, and the satisfaction of many specifications, we developed methods tailored to cope with these issues.

We can then say that, even though the title of this book refers to a very specific domain of application, it is intended for people with interests and backgrounds both in computer science and in electronics: computer scientists interested in acquiring basic skills in Evolutionary Computation; Evolutionary Computation researchers interested in learning new techniques; and, naturally, electronic engineers interested in learning about this versatile tool for Computer Aided Design (CAD) of analog and digital electronic circuits. Furthermore, we believe that this book will attract engineers and computer scientists interested in biology, particularly in the field of natural evolution. We try to be very precise when describing the biological concepts modeled by the proposed algorithms.

The first unit of this book consists of a single and introductory chapter, whose objective is to define and describe the field of Evolutionary Electronics. This chapter presents the main challenges and problems in electronic design, describing some forthcoming technologies as well. We proceed by justifying the motivation that points towards the use of Evolutionary Algorithms as a tool that can handle future challenges in this area. This chapter continues by presenting a description of the setup involving experiments in Evolutionary Electronics, contrasting two different experimental environments: *evolution through simulation* and *real-time evolution*. We finish by presenting a brief survey on the current state of the art of Evolutionary Electronics.

The second unit of this book describes the evolutionary computation paradigm, which is embedded in a class of algorithms that seeks for inspiration in biological phenomena. Evolutionary computing methods are based on natural evolution principles. Its most important counterpart, artificial neural networks, takes the brain computational paradigms as its main source of inspiration. In **Chapter 2** the authors describe the history and development of evolutionary computation and review the basic flow and operations of this class of algorithms. This chapter includes an introduction to natural evolution as well, providing the reader with a background on the area that lent the main ideas used in evolutionary computation. This chapter provides a description of the main evolutionary algorithms found in the literature: evolutionary strategies; evolutionary programming; genetic algorithms; and genetic programming. **Chapter 3** complements our theoretical study, looking into some advanced topics that are currently being investigated by the Evolutionary Computation community: multi-objective optimization; variable length representation systems; speciation; and deception. These selected topics are tailored to the purpose of this book, which is to show evolutionary computing as a tool to be applied to *real problems*.

The third unit encompasses a number of Evolutionary Electronics applications. We report several applications, ranging from the optimization of BiCMOS operational amplifiers to the synthesis of PID (Proportional-Integral-Derivative) controllers. **Chapter 4** focuses on the optimization of analog and digital VLSI integrated circuits. Many different problems are described, such as the optimization of operational amplifiers using different technologies; placement and routing in digital chips; and transistors' size optimization for digital logic gates. **Chapter 5** describes applications in the synthesis of basic analog electronic circuits, commonly used as building

blocks for more complex systems. This chapter includes the synthesis of passive and active filters; synthesis of operational amplifiers based on bipolar and synthesis of a Digital to Analog converter. Conversely, **Chapter 6** describes applications in the synthesis of digital circuits, including the design of combinational and sequential circuits; the evolutionary synthesis of logic gates based on bipolar transistors; and the synthesis of digital filters. **Chapter 7** presents the topic of evolutionary circuit synthesis through the use of reconfigurable chips. We introduce the basic concepts of reconfigurable circuits, emphasizing Field Programmable Gate Arrays (FPGAs) and Field Programmable Analog Arrays (FPAAs). We then present Evolutionary Electronics applications performed over these platforms, and finish by introducing a more advanced topic, *Virtual Computing*. **Chapter 8** goes one step ahead, and describes the evolutionary design of more complex systems. We address the following topics: evolutionary synthesis of Gm-C Filters; synthesis of analog controllers in general and of PID controllers; the evolutionary synthesis of antennas, and fault tolerant evolutionary electronics. As a whole, the third unit provides the reader with a variety of applications, covering the state of the art in Evolutionary Electronics.

Unit IV of this book draws conclusions over the applications described and makes a sketch of the future of evolutionary computation and its applications in electronics.

Finally, the appendixes provide an introduction to four important related subjects:

- Appendix A: MOS Transistors
- Appendix B: Switched Capacitor (SC) Circuits
- Appendix C: Gm-C Filters
- Appendix D: An Introduction to Nano Electronics

The authors believe that evolutionary computation will play an important role in general design processes, in a world where the appearance of more advanced technologies increases the complexity of optimization and synthesis problems. As a consequence, Evolutionary Electronics may turn into a key technology in the following years.

ACKNOWLEDGMENTS

This book was written and prepared at ICA (Applied Computational Intelligence Laboratory) at PUC-Rio (Pontifical Catholic University), Rio de Janeiro, Brazil. Many people at ICA helped to make this book possible. We would like to express our deep gratitude to them in this effort.

Firstly, thanks are due to people who contributed to this book in different sections with the results of their research work; these are (in alphabetical order): Rodrigo M.L.A. Costa, Maíra Ferreira de Noronha, Cristina Santini, Hélio Takahiro Sinohara,and Moisés Henrique Szwarcman.

We are also deeply indebted to our undergraduate and postgraduate students Carlos Augusto Louzada Costa, Paulo Henrique Dargam, Lucas Ferraz, Laércio Brito Gonçalves, and Karla Tereza Figueiredo Leite who spent days of work formatting text, drawing figures, correcting typing, and so on.

We would like to extend our thanks to the editors for their support and comprehension.

We also would like to acknowledge a series of grants received from Brazilian funding agencies: CAPES, CNPq, and FAPERJ, which allowed us in many ways to write this book.

In this book we have included many citations from other published work, as well as parts of our own published papers. We would like to acknowledge all permissions received from their editors to use their material in this book.

Finally, we would like to thank our institution PUC-Rio University and the Department of Electric Engineering for their support.

Contents

UNIT I - INTRODUCTION

UNIT II - EVOLUTIONARY COMPUTATION

UNIT III - EVOLUTIONARY ELECTRONICS

UNIT IV - CONCLUSIONS

CHAPTER 1

Introduction to Evolutionary Electronics

This chapter introduces Evolutionary Electronics. We start by providing a formal definition of Evolutionary Electronics, as well as the main concepts underlying this research field. Following that, we explore the main motivation for investigating this new approach to circuit and systems design: the idea of abstaining from human knowledge to design circuit and systems by interpreting electronic design as a search task. We finish this chapter presenting an overview of the state of the art of the area, listing the main works that have been realized by the research community around the world.

1.1 EVOLUTIONARY DESIGN OF ELECTRONIC CIRCUITS: A PROSPECT OF THE AREA

In September of 1997 some researchers, coming from different universities in Europe, gathered together at the University of Napier, in Scotland, with the objective of formalizing the new area of research they had been involved with for the past recent years, which involved *Evolutionary Computation* and *Electronics.* Stimulated by the promising results they and other researchers in the US and Japan have achieved, they decided to meet with the following agenda: give a formal name to this area of research; define which applications would fall into the domain of this new area; identify the main groups that have been working in this new field; compile the most important articles published so far; and organize a report on the results achieved so far.

They came up with the name of ***Evolutionary Electronics***, an area that would cover all the applications involving the use of Evolutionary Computation in electronic systems' design. For those readers who are not familiar with Evolutionary Computation, the next two chapters of this book will provide the basic definitions and advanced concepts related to Evolutionary Computation. For now, we will just say that Evolutionary Computation uses search tools, encompassing a particular class of algorithms which employs some aspects of natural evolution as metaphors. These algorithms, called Evolutionary Algorithms (EAs), have been widely applied to complex optimization problems. For the scope of this chapter we can simply think of Evolutionary Algorithms as search tools that sample many points of a particular search space, searching for a point that satisfies a certain specification. We list below two main aspects related to EAs:

- EAs typically sample many possible solutions (up to 10^8) in order to find one that satisfies the specifications, although the number of solutions sampled is often negligible compared to the huge size of the search space (up to 10^{500}).

- The EA sampling is not performed in a random way; instead, natural evolution principles, as mentioned above, are used to drive the search process.

We will develop these concepts in detail in the next unit of this book.

Even though most of the researchers in that meeting have been investigating Evolutionary Electronics only for the last couple of years, the first reported applications actually appeared in the middle 80's, when evolutionary algorithms were applied to digital chips optimization, in tasks such as placement and routing[1]. However, in the early 90's, two facts contributed to the expansion of this research field:

- The idea of using these Evolutionary Algorithms not only to optimize digital chips layout, but to accomplish the whole process of circuit design, including devising the circuit topology from scratch

- The development of a new class of integrated circuit devices, called Field Programmable Gate Arrays (FPGAs). These devices can be quickly reconfigured by software, being able to implement a huge variety of digital circuits.

Let us first draw some comments on the first item mentioned above. The work of Louis and Rawlins[2] was one of the first to introduce the idea of using Evolutionary Algorithms as tools to perform *structure design*. Particularly, they presented the notion of structure design for digital circuits: the EA was used to conceive arrangements of digital gates that could solve a specific problem, such as the parity function. This idea was highly original, since the evolutionary algorithm performed thorough automatic circuit design from scratch, i.e., no human knowledge was needed. This was radically different from the previous approaches reported in the literature, where EAs were used only to optimize VLSI layouts resultant from human designed circuits. Additionally, the idea of using a computational algorithm to achieve a circuit design from scratch contrasted with the procedure followed by most CAD (Computer Aided Design) tools for circuit design, which were based on human heuristics and constraints.

The development of modern reconfigurable chips also played an important role in the development of Evolutionary Electronics. In particular, FPGA devices represent a technological advance in comparison with the early reconfigurable devices, since the former includes a higher density of logic gates integrated through VLSI technology, and also display a lower reconfiguration time. Although FPGAs were originally conceived for other purposes, their use as *evolvable chips* was proposed in 1993 by H. De Garis.[3] FPGAs play the role of evolvable chips when they are automatically reconfigured by an evolutionary algorithm. In this case, the search process performed by the Evolutionary Algorithm takes place over the FPGA platform: the evolutionary algorithm instantiates different circuits on the reconfigurable device until one that complies to a set of specifications is possibly found. This book has

an entire chapter dedicated to evolution of circuits in reconfigurable chips such as FPGAs and their analog counterparts, the Field Programmable Analog Arrays (FPAAs).

According to the kind of electronic design, Evolutionary Electronics can be classified into three categories, *digital*, *analog,* or *mixed-signal*.

Applications of Evolutionary Electronics on digital design are usually the most challenging ones due to two reasons: the greater complexity of the search space;[4] and the existence of efficient CAD design tools dedicated to digital circuits. The first reason stems from the intrinsic nature of digital design which will be discussed later in this book. The second reason refers to the fact that it is difficult to develop new CAD tools that provide competitive performance when compared with the already existing ones. Nevertheless, despite requiring more research effort, many works have already been published where Evolutionary Electronics outperformed human designed circuits. Additionally, it is a general belief that, as digital design requirements become more complex, existing CAD tools may fail to produce satisfactory solutions. Another important reason for developing research in evolutionary design of digital circuits is the importance of digital technology in modern electronics.

Analog circuit design, on the other hand, is much more amenable for evolutionary techniques. The search space associated with analog design problems is smoother, being more adequate to be sampled by Evolutionary Algorithms. Contrasting with digital design, there is no solid set of design rules or procedures for analog circuit synthesis. As a consequence, one usually has to rely on the experience and even the intuition of the engineer. In addition, CAD tools for analog design are not so common as the ones for digital hardware. One may always argue that analog design has been experiencing a decreasing level of importance in comparison with digital design; however, modern integrated chips are increasingly needing analog circuitry, mainly at their interface with the outside world, which is analog. Furthermore, Evolutionary Algorithms may yield analog circuits that outperform digital hardware in areas where digital technology was traditionally superior. This is summarized in Stoica et al.[5] :

> *One reason is that the potential for analog processing is much greater than what is exploited today. Analog circuitry has advantages in cost, size and power consumption (compared to digital), and can directly process signals that are continual in time and amplitude, albeit at a finite resolution. Even a single transistor has many functions that can be exploited in computation functions such as generation of square, square-root, exponential and logarithmic functions. Other functions include voltage-controlled current sources, analog multiplication of voltages, and short term and long term analog storage. The basic combinations of transistors offer a rich repertoire of linear and nonlinear operators available for local and collective analog processing. Using evolution, the*

benefits of analog processing can be exploited, while its disadvantages are reduced/eliminated.

Naturally, mixed signal evolutionary electronics refer to the evolution of circuits with both analog and digital parts.

Another important taxonomy of Evolutionary Electronics refers to the use of circuit simulators or reconfigurable chips as platforms for the search process. The former methodology is called extrinsic evolution, whereas the latter is called intrinsic evolution, real-time evolution, or Evolvable Hardware (EHW). Table 1.1 below presents a comparative analysis between extrinsic and intrinsic evolution.

Table 1.1 Comparison between extrinsic and intrinsic evolution of electronic circuits

Feature	***Extrinsic***	***Intrinsic***
Time	application dependent	application dependent
Novel Circuits	possible to generate	possible to generate
Experiment Complexity	smaller	higher
Search Space	smaller	larger
Silicon Properties	Partially Considered	Fully Considered
Reliability	lower	higher
Robustness	higher	lower

According to Table 1.1, there are many aspects for the comparison of extrinsic and intrinsic evolution of electronics circuits. Depending on the particular application, either the extrinsic or the intrinsic methodology may suit better.

In terms of *time*, it is not yet possible to establish which method will suit better, i.e., the one that would provide the lowest time to execute the evolutionary algorithm. In most cases, the real-time behavior of an electronic circuit occurs in a time-scale much smaller than the one achieved by simulators, a fact that benefits intrinsic evolution. This happens because simulations often rely on the solution of differential equations, particularly in the case of analog circuits. However, when the length of time to be analyzed is higher (in the range of seconds) due to the slow circuit dynamics, then simulators are more appropriate, since they can obtain the circuit's response in less than one second. There is also a great variability in the duration of the intrinsic experiments, depending on how the configurable chip interfaces with the host computer. The time to download a particular circuit and to obtain its output can sometimes increase the duration of the experiment.

Both intrinsic and extrinsic experiments can produce *novel circuit designs* compared with the ones found in the literature. In both cases, it is not necessary to constrain the search process to sample only well-known topologies; instead, the designer is usually interested in departing from these constraints in order to achieve novel circuits.

It is usually more *complex* to run an intrinsic experiment than an extrinsic one. In the former, the hardware interface between the host and the reconfigurable chip sometimes has to be implemented by the user. In the latter, a simple software interface between the Evolutionary Algorithm and the circuit simulator, such as SPICE[6] or

SMASH,[7] has to be implemented. Furthermore, in the case of reconfigurable chips, some illegal circuit configurations generated by the evolutionary algorithm may damage the circuitry.

The *search space* is often larger in the case of intrinsic experiments. As it will be explained further in this book, reconfigurable chips' circuits are encoded through large binary strings, resulting in very large search spaces. On the other hand, in the case of extrinsic experiments, the search space size can be controlled by the user, through optimized representations. Evolutionary Algorithms are particularly targeted to handle large search space; however, when the search space becomes exaggeratedly large, even the Evolutionary Algorithm will barely sample a very small fraction of it, damaging the evolutionary process.

Another important property of Evolutionary Electronics refers to the exploration of silicon properties. In principle, when an electronic circuit is being evolved using a programmable chip, novel and strange topologies, that still conform with the user's specifications, may result from the evolutionary process. In order to follow the specifications, these novel circuits may be taking into account the thermal and electromagnetic properties of the silicon, instead of using only its ordinary electronic properties. Additionally, parasitic capacitances may also play an important role in the circuit's behavior, which can not be evaluated in the extrinsic process. In extrinsic evolution, silicon properties can not be explored in the same extent, since simulators do not model so accurately the components' physical characteristics.

As explained earlier, both extrinsic and intrinsic methods can explore novel designs. However, due to the fact that simulators do not have such accurate models for some subtle semiconductors' properties, it is sometimes possible that novel circuits produced extrinsically may behave differently when implemented in real silicon compared to the behavior in the simulation. Thus, extrinsic circuit evolution is not as reliable as the intrinsic method. Intrinsic evolution employs no models or abstractions for the semiconductors[8] there is no difference between design and implementation. In the near future, when simulators incorporate more accurate models of semiconductors' devices and advanced CPUs speed up the simulation process, we can expect that extrinsic methods will be more reliable.

Finally, there is a drawback associated with the exploration of subtle silicon properties promoted by real-time evolution: the lack of *robustness* of the circuits. Circuits that use uncommon physical properties will tend to work only within those particular conditions in which the evolutionary process took place: temperature, power supply, external electromagnetic influences, and the particular programmable chip utilized. As observed by Thompson et al.,[9] an intrinsically evolved circuit may display different behaviors when instantiated in different programmable chip samples.

We can then summarize the most notable features of circuits' evolution observed in a lesser or greater extension in both, intrinsic and extrinsic evolution:

- Potential to find *novel circuits*
- The possibility to find *new design rules* from the novel circuits obtained

- Evolutionary methods can contemplate a larger set of design specifications compared to human design techniques
- Evolutionary systems have been able to achieve competitive circuits when compared with the state of the art in electronics
- As evolutionary tools are more appropriate to analog design, we can even expect that this will produce a new trend towards the research in analog circuits

Among the features described above, the third one should be emphasized. Nowadays, it is important to achieve electronic circuits that conform to many specifications: good performance; low area; low power consumption; speed; and, in some cases, fault tolerance requirements. Conventional design methods are usually not appropriate when many specifications are considered. As it will be described later in this book, evolutionary algorithms can incorporate methodologies that process many different design objectives.

1.2 ELECTRONIC CIRCUIT DESIGN: A SEARCH TASK

Evolutionary Electronics considers a new concept for automatic design of electronic systems: instead of using human conceived models, abstractions, and techniques, it employs search algorithms to develop good designs. This idea is summarized in Thompson et al.:[9]

> *Imagine a design space where each point in that space represents the design of an electronic circuit. All possible electronic circuits are there, given the component types available to electronics engineer, and the technological restrictions on how many components there can be and how they can interact. In this metaphor, we loosely visualize the circuits to be arranged in the design space so that 'similar' circuits are close to each other.*
>
> *The design space is vast. There are oscillators and filters, finite-state machines, analogue computers, parallel distributed systems, Von Neumann computers, and so on. These nestle amongst the majority of circuits for which a use will never be found.*

Evolutionary Electronics deals with a huge search space, requiring, therefore, powerful search techniques to handle the task. Naturally, when the search space is very large, only random search has some chance to succeed. Hence, this new approach to circuit design does not resume choosing a design problem and applying

a search technique. Instead, some procedures have to be followed:

- The search space sampled by the algorithm must have its size limited. Although it is important to allow the sampling of a wide variety of topologies, some criteria should be chosen to control the number of possible solutions.

- It is usually necessary to adapt the search techniques to the particularities of the design problem.

The search space size is a very subtle issue: it should be large enough to include a good variety of novel circuit topologies; nevertheless, if the design space increases without restriction, the chances to find a good solution are very small. This is a very important issue for the search technique applied in this book: Evolutionary Algorithms. These algorithms suit well to sample large design spaces, performing better than standard optimization techniques. Nonetheless, for very large search spaces, even EA presents its limitations. Evolutionary Algorithms are adequate to the kind of problems we will be analyzing in this book. We can restrict the search space size and simultaneously keep its diversity of designs. The search space is usually limited by setting a maximum size for the circuits to be sampled.

Another important issue is the inclusion of special techniques into the search tool in order to successfully find a circuit that conforms with all specifications. It must be remarked that these special techniques are not related to some kind of previous knowledge on the electronic design; instead, the approach proposed in this book is to improve the search tools by introducing new techniques to handle the specific properties of the electronic design process, such as its multi-objective nature and the unpredictability of the size of the solution. This idea will be clarified along this book.

Evolutionary Algorithm also provides the possibility to aggregate powerful paradigms in its methodology. These new paradigms can greatly improve the performance of EAs applied to electronic design. Even though we believe in the better suitability of EAs as search tools for electronic design, we do not preclude the possibility of successful application of other search techniques, such as HillClimbing[10] and Simulated Annealing.[11]

After this discussion one may think: Why would we be interested in using search algorithms to perform electronic design? Is not the size of the search space to be sampled a drawback that we would not have to face using conventional techniques? If we go back to the last section, when we summarized the most notable features of circuits' evolution, some answers to this question can be found: the possibility to find new circuit designs and the capacity to contemplate a larger set of specifications, among others. We can add two more motivations:

- The increasing level of complexity of electronic technology and performance specifications may render conventional techniques useless to carry out the forthcoming generation of circuit design.

- The constant increase in the microprocessors' speed will partially alleviate the drawback of the design space size.

We believe that the following remarks, found in Thompson et al.,[9] summarize everything that has been said so far in this chapter:

> *Conventional design methods can only work within constrained regions of the design space. Most of the whole search space is never considered.*
>
> *Evolutionary Algorithms can explore some of the regions of the design space that are beyond the scope of conventional methods. In principle, this raises the possibility that designs can be found there that are in some sense better.*

1.3 A BRIEF SURVEY OF EVOLUTIONARY ELECTRONICS

We will now list some relevant works on Evolutionary Electronics (see Table 1.2). This is not a complete list of research works in the area; there are many other no less important works that have not been mentioned here. Our intention is just to identify the research applications that started innovative trends in the area. We also remark that, in many cases, the name Evolvable Hardware is also used as a synonym for Evolutionary Electronics. Throughout this book, however, we will only use the term Evolvable Hardware as a specific case of Evolutionary Electronics where the evolutionary system is carried out over a reconfigurable platform.

Table 1.2 Relevant works on evolutionary electronics

Date	Authors	Application
1991	Louis and Rawlins[2]	Evolution of basic digital functions
1993	H. De Garis[3]	Introduction of the concept of Evolvable Hardware
1995	Higuchi et al.[12]	Evolution of digital circuits for pattern recognition
1995	Thompson et al.[13]	Evolution of a hardware sensorimotor control structure
1995	Hemmi et al.[14]	Use of Hardware Description Language to evolve circuits
1995	Grimbleby[15]	Automatic Analog Network synthesis using EAs
1995	Mange et al.[16]	Reconfigurable chips with Self-Repair and Self-Reproducing properties
1996	Thompson[17]	First intrinsically evolved circuit using an FPGA chip
1996	Sanchez[18]	Overview on biological inspiration for innovative hardware design
1996	Koza et al.[19]	Evolution of low-pass filter and bipolar transistor amplifiers
1996	Nussbaum et al.[20]	Digital hardware design based on biological organisms features
1996	Liu et al.[21]	Evolution of digital circuits for ATM cell scheduling
1996	Salami et al.[22]	Evolution of digital circuits for data compression
1996	Murakawa et al.[23]	Evolution of digital circuits for equalization of communication channels
1996	Keymeulen et al.[24]	Evolution of a digital circuit for a robot navigation system
1997	Miller et al.[25]	Evolution of novel arithmetic digital circuits
1998	Layzell et al.[26]	Introduction of a reconfigurable analog tool for Evolvable Hardware
1998	Flockton and Sheenan[27]	Intrinsic evolution of analog circuits
1998	Murakawa et al.[29]	Evolution of an analog filter targeted for a commercial application

1998	Zebulum et al.[30]	Evolution of a digital circuit for CPU control
1998	Koza et al.[31]	Evolution of analog circuit for control applications
1998	Thompson et al.[32]	First results on the intrinsic evolution of robust circuits
1998	Hamilton et al.[33]	Introduction of a reconfigurable analog tool for Evolvable Hardware
1999	Miller et al.[34]	Evolution of digital filters
1999	Stoica et al.[5]	Evolution of CMOS analog circuit on a novel reconfigurable chip
1999	Zebulum et al.[35]	Multiple-objectives' evolution of active filters
1999	Lohn et al.[36]	Multiple-objectives' evolution of analog circuits
1999	Linden and Altshuler[37]	Survey report on the evolution of wire antennas

REFERENCES

[1] Cohoon, J.P. and Paris, W.D., *Genetic Algorithms in Engineering Systems*, The Institute of Electrical Engineers, London, 1997.

[2] Louis, S.J. and Rawlins, J.E., Designer genetic algorithms: genetic algorithms in structure design, ICGA-91, in *Proc. of the Fourth International Conference on Genetic Algorithms*, Belew, R.K., Booker, L.B., and Kauffman, M., Eds., 1991, 53.

[3] DeGaris, H., *Evolvable Hardware: Genetic Programming of a Darwin Machine, in Artificial Neural Nets and Genetic Algorithms*, Albretch, R.F., Reeves, C.R., and Steele, N.C., Eds., Springer-Verlag, New York, 1993.

[4] Vassilev, V.K., Fogarty, T.C., and Miller, J.F., Information characteristics and the structure of landscapes, in *Evolutionary Computation,* 8, 1, MIT Press, Cambridge, 2000, 31.

[5] Stoica, A., Keymeulen, D., Tawel, R., Salazar-Lazaro, C., and Li, W., Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital CMOS circuits, in *Proc. of the First NASA DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D. and Lohn, J., Eds., IEEE Computer Press, 1999, 76.

[6] Quarles, T., Newton, A. R., Pederson, D. O., and Sangiovanni-Vincentelli, A., SPICE3 Version 3f5 User's Manual, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California , 1994.

[7] *SMASH User and Reference Manual*, Dolphin Integration, France, 1993.

[8] Thompson, A., Harvey, I., and Husbands, P., Unconstrained evolution and hard consequences, in *Towards Evolvable Hardware: An International Workshop*, Lausanne, Swiss, 1995, 2. Chapter of book: *Towards Evolvable Hardware: The evolutionary engineering approach*, Sanchez, E. and Tomassini, M., Eds., Springer-Verlag LNCS 1062, 1996, 136.

[9] Thompson, A. and Layzell, P., Analysis of unconventional evolved electronics, *Communications of the ACM*, Yao, X., Ed., 42, 4, 1999, 71.

[10] Winston, P.H., *Artificial Intelligence*, 3rd ed., Addison-Wesley, Reading, MA, 1992.

[11] Kirkpatrick, S., Gelatt, C.D., Jr., and Vecchi, M.P., Optimization by simulated annealing, *Science*, 220, 1983, 671.

[12] Higuchi, T. et al., Evolvable hardware and its applications to pattern recognition and fault-tolerant systems, in *Towards Evolvable Hardware: An International Workshop*, Lausanne, Swiss, 1995, 2. Chapter of book: *Towards Evolvable Hardware: The evolutionary engineering approach*, Sanchez, E., and Tomassini, M., Eds., vol. 1062, LNCS, Springer-Verlag, 1996, 118.

[13] Thompson A., Evolving electronic robot controllers that exploit hardware resources, CSRP 368, in: Advances in artificial life, *Proceedings of the 3rd European Conference on Artificial Life (ECAL95)*, Springer-Verlag Lecture Notes in Artificial Intelligence, 929, 1995, 640.

[14] Hemmi, H., Mizoguchi, J., and Shimonara, K., Development and evolution of hardware behaviours, in *Towards Evolvable Hardware: An International Workshop*, Lausanne, Swiss, 1995, 2. Chapter of book*: Towards Evolvable Hardware: The evolutionary engineering approach*, Sanchez, E., and Tomassini, M., Eds., vol. 1062, LNCS, Springer-Verlag, 1996, 250.

[15] Grimbleby, J. B., Automatic analog network synthesis using genetic algorithms, in *Proc. of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems* (GALESIAS - 95), England, 1995, 53.

[16] Mange, D. et al., Embryonics: A New Family of Coarse-Grained Field Programmable Gate Array with Self-Repair and Self-Reproducing Properties, in *Proc. of the First Workshop on Evolvable Hardware, Towards Evolvable Hardware: The Evolutionary Engineering Approach*, Sanchez, E., and Tomassini, M., Eds., vol. 1062, LNCS, Springer-Verlag, 1996.

[17] Thompson, A., Silicon Evolution, in *Proceedings of Genetic Programming* 1996 (GP96), J.R. Koza et al., Eds., MIT Press, Cambridge, 1996, 444.

[18] Sanchez, E., Mange, D., Sipper, M., Tomassini, T., Perez-Uribe, A., and Stauffer, A., Phylogeny, ontogeny and epigenesis: three sources of biological inspiration for softening hardware, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259, LNCS, Springer-Verlag, 1996, 35.

[19] Koza, J.R., Bennett III, F.H., Andre, D., and Keane, M. A., Toward evolution of electronic animals using genetic programming. Artificial Life V, in *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, 1996, 327.

[20] Nussbaum, P., Marchal, P., and Piguet, C., Functional organisms growing on silicon, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259, LNCS, Springer-Verlag, 1996, 139.

[21] Liu, W., Murakawa, M, and Higuchi, T., ATM cell schedule by function level evolvable hardware, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259, LNCS, Springer-Verlag, 1996, 180.

[22] Salami, M., Murakawa, M., and Higuchi, T., Data compression based on evolvable hardware, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259, LNCS, Springer-Verlag, 1996, 169.

[23] Murakawa, M., Yoshizawa, S., and Higuchi, T., Adaptive equalization of digital communication channels using evolvable hardware, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259. LNCS, Springer-Verlag, 1996, 379.

[24] Keymeulen, D., Durantez, M., Konaka, K., Kuniyoshi, Y., and Higuchi, T., Evolution of a digital circuit for a robot navigation system, in *Proc. of the First International Conference on Evolvable Systems: From Biology to Hardware* (ICES96), Tsukuba, Japan, Goos, G., Hartmanis, J., and Leeuwen, J. van, Eds., vol. 1259, LNCS, Springer-Verlag, 1996, 195.

[25] Miller, J. F., Thomson, P., and Fogarty, T., Designing electronic circuits using evolutionary algorithms, arithmetic circuits: a case study, in *Genetic Algorithms Recent Advancements and Industrial Applications*, Quagliarella, D., Periaux, J., Poloni, C., and Winter, G., Eds., Wiley & Sons, 1997, New York, chap. 6.

[26] Layzell, P., A new research tool for intrinsic hardware evolution, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 47.

[27] Flockton, S. J. and Sheehan, K., Intrinsic circuit evolution using programmable analog arrays, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 144.

[28] Zebulum, R.S. Titulo, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998.

[29] Murakawa, M., Yoshizawa, S., Adachi, T., Suzuki, S., Takasuka, K., Iwata, M., and Higuchi, T., Analogue EHW chip for intermediate frequency filters, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 134.

[30] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., Evolutionary systems applied to the synthesis of a CPU controller, in *Proc. Lectures Notes in Artificial Intelligence*, Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL98), Canberra, Australia, Ed., Springer-Verlag, 1998.

[31] Koza, J., Bennett III, F.H., Andre, D., Keane, M.A., and Dunlap, F., Automated synthesis of analog electrical circuits by means of genetic programming, *IEEE Transactions on Evolutionary Computation*, 1(2), 1998, 109.

[32] Thompson, A., On the automatic design of robust electronics through artificial evolution, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 13.

[33] Hamilton, A., Papathanasiou, K., Tamplin, M.R., and Brandtner, T., Palmo: field programmable analog and mixed-signal VLSI for evolvable hardware, in *Proc. of the Second International Conference on Evolvable Systems: From Biology to Hardware* (ICES98), Lausanne, Switzerland, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 335.

[34] Miller, J., On the filtering properties of evolved gate arrays, in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D., and Lohn, J., Eds., Pasadena, CA, 1999, 225.

[35] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., Artificial Evolution of Active Filters: A Case Study, in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D., and Lohn, J., Eds., Pasadena, CA, 1999, 66.

[36] Lohn, J.D. and Haith, G.L., A Comparison of Dynamic Fitness Schedules for Evolutionary Design of Amplifiers, in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D., and Lohn, J., Eds., Pasadena, CA, 1999, 87.

[37] Linden, D.S. and Altshuler, E.E., Evolving wire antennas using genetic algorithms: a review, in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D., and Lohn, J., Eds., Pasadena, CA, 1999, 225.

CHAPTER 2

Introduction to Evolutionary Computation

This chapter starts by introducing the basic concepts of natural evolution, from which evolutionary computation borrowed its main principles. We then provide a detailed description of the components of Evolutionary Algorithms (EAs), i.e., the representation, fitness evaluation, and main operators. Finishing this chapter we describe the main evolutionary algorithms reported in the literature: Evolutionary Programming; Evolutionary Strategies; Genetic Algorithms; and Genetic Programming.

This chapter provides the reader with the basic ideas behind evolutionary computation. The authors do not assume the readers have any kind of previous knowledge of the area. However, we encourage those who already are acquainted with this field not to skip this chapter, since we attempt to give a broader view of the field, including the most important topics of research in evolutionary computation. The first section of this chapter provides an overview of the area of natural evolution. The basic concepts of the area are presented in a very simple way, which makes the comprehension of the basic ideas incorporated by the evolutionary algorithms very easy. The second section of this chapter describes the basic components of evolutionary algorithms, encompassing the representation, fitness evaluation function, and the main operators. Whenever convenient, we draw analogies with the biological counterparts observed in nature. The third section reviews the main evolutionary algorithms proposed in the literature, from the very first attempts at the end of the 50's, to the widespread evolutionary algorithms currently used nowadays, such as genetic algorithms and genetic programming.

2.1 BASIC CONCEPTS OF NATURAL EVOLUTION

This section begins by analyzing the basic ideas constituting the biological evolution theory,[1] which were compiled in the 19th century by the naturalist Charles Darwin. We then proceed with the introduction of some basic concepts of molecular genetics, which is important to understand the main evolutionary mechanisms. Next, we describe the important role of natural selection, recombination, and mutation in the theory of natural evolution. We end this section by listing some facts that give evidence of the evolution of species.

2.1.1 The Evolutionary Hypothesis

In a biological sense, the word *evolution* encompasses the processes that take place between generations within a population of species. Contrasting with the initial belief that species have been fixed in form and created separately, the evolutionary theory states that all species have evolved from a common ancestor.

The foundations of this idea have been formulated by the naturalist Charles Robert Darwin[2] (1809-1882). He drew the most important conclusions of his work after a voyage to the Galapagos Islands. By observing, among other species, the birds of these islands, he arrived at several hypotheses that would build his theory:

- Due to the struggle for existence observed in nature, the forms that are better adapted to survive will leave more offspring and automatically increase in frequency from one generation to the next.

- Forms of a particular species that adapt better to environmental changes through time will increase in frequency, while poorly adapted forms will decrease in frequency.

- In the face of the above observations a new theory, based on the change in species and in the formation of new species, could be derived.

The principle of struggle to survival played an important role in Darwin's hypothesis. This principle is known as *natural selection*, stating that "*given the conditions of limited resources and excess of fecundity, members of a population and of different species will compete to survive"*.[1] Nonetheless, certain conditions must be observed for natural selection to operate. Among these conditions, the most important is the requirement of variation in the *fitness* of organisms. In biology, fitness means the average of the number of offspring left by an individual relative to the number of offspring left by the average member of the population. Natural selection favors higher fitness individuals.

The famous example of the variation in frequency of melanic and peppered moths in the United Kingdom, during the industrial revolution, proved that natural selection could take place in a small length of time. While peppered moths were predominant before the industrial revolution, the melanic forms turned out to outnumber the peppered ones after the industrial revolution, since they were better camouflaged against predatory attack in industrial regions. So, as it is stated in Ridley:[1]

> *... If the process that operated in the nineteenth century in a single species of moth had been continued for the thousands of millions of years since life originated, much larger evolutionary changes could be accomplished. Indeed, one version of evolutionary theory maintains that the abstract process – natural selection – that caused the particular change in the peppered moth population is also responsible, in a general way and over a much grander scale, for the whole diversification of life from a simple common ancestor about 3.5 x 10^9 years ago.*

Even though many of Darwin's contemporaries accepted the idea of evolution, his hypothesis of natural selection was not accepted. Furthermore, Darwin's theory lacked a theory of heredity. We will then review some basic concepts of molecular and Mendelian genetics, which in some way complemented Darwin's hypothesis, giving rise to Neo-Darwinism.

2.1.2 Molecular Genetics

In order to understand how evolution is accomplished in natural systems, it is necessary to briefly present some concepts of molecular genetics. The molecule called DNA (deoxyribonucleic acid) provides the physical mechanism of heredity in all living creatures, containing the information needed to build a body. The DNA molecule is made up of a sequence of units, called nucleotides, which consist of a phosphate and a sugar with a basis attached. DNA molecules exist inside almost all of the cells of a body and they are physically carried in structures called chromosomes.

Let us clarify the means whereby the DNA encode the information to build a body. Regarding the human DNA, this question is addressed in Ridley[1] in the following way:

> *The DNA in an individual human cell contains about 3 x 10^9 nucleotide units. This total length can be divided into genes and other kinds of DNA. Some genes lie immediately next to neighboring genes; others are separated by regions of varying length. Each gene contains the information that codes for a particular protein.*

Since we can almost affirm that bodies are built from proteins, then the DNA has a kind of receipt to construct a body. A concise description of how the DNA encodes for proteins is again given in Ridley:[1]

> *... the sequence of nucleotides in a gene specifies the sequence of amino acids in the protein. DNA contains four types of nucleotides that differ only in the base part of the nucleotide unit: the sugar and phosphate groups remain in the same four types. The four types of nucleotides are called adenine (A), cytosine (C), guanine (G) and thymine (T). Adenine and guanine belong to the chemical group called purines, while cytosine and thymine are pyramidines. In the double helix, an **A** nucleotide in one strand always pairs with a **T** nucleotide in the other; and a **C** always pairs with a **G**.*
>
> *... a triplet of basis encodes for one amino acid; the nucleotide triplet for an amino acid is called a codon ... The relation between the triplet and the amino acid is called genetic code.*

As explained above, the DNA is structurally organized in a double strand, where complementary pairs of nucleotides are located. Figure 2.1 roughly depicts the DNA structure for a hypothetical sub-sequence of nucleotides. Although the four nucleotides can be arranged in 64 (4 x 4 x 4) distinct codons, there exist only 20 amino acids. Hence, there is not a one-to-one relation between the nucleotide triplet and the amino acid. This relation, called genetic code, is summarized in Table 2.1. This table includes the name of the 20 amino acids, as well as the so-called stop codons, which signal the end of a gene.

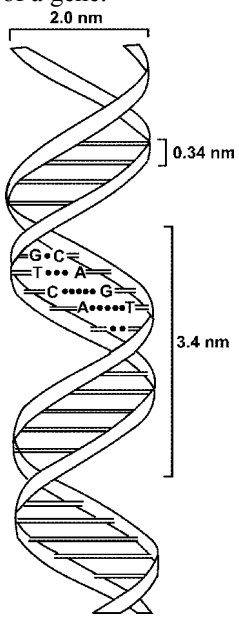


Figure 2.1 The structure of DNA.[1]

Table 2.1 The genetic code[1]

First Base in the Codon	Second Base in the Codon				Third Base in the Codon
	U	C	A	G	
U	Phenylalanine	Serine	Tyrosine	Cysteine	U
	Phenylalanine	Serine	Tyrosine	Cysteine	C
	Leucine	Serine	Stop	Stop	A
	Leucine	Serine	Stop	Tryptophan	G
C	Leucine	Proline	Histidine	Arginine	U
	Leucine	Proline	Histidine	Arginine	C
	Leucine	Proline	Glutamine	Arginine	A
	Leucine	Proline	Glutamine	Arginine	G
A	Isoleucine	Threonine	Asparagine	Serine	U
	Isoleucine	Threonine	Asparagine	Serine	C
	Isoleucine	Threonine	Lysine	Arginine	A
	Methionine	Threonine	Lysine	Arginine	G
G	Valine	Alanine	Aspartic acid	Glycine	U
	Valine	Alanine	Aspartic acid	Glycine	C
	Valine	Alanine	Glutamic acid	Glycine	A
	Valine	Alanine	Glutamic acid	Glycine	G

Let us look into the process of protein synthesis in more detail. This process is an essential cell function and it is brought about in two steps, ***transcription*** and ***translation***. The transcription process occurs in the cell nucleus, and it refers to the synthesis of an intermediate molecule, the messenger RNA or mRNA. The mRNA is a single stranded molecule transcribed from the DNA. The main difference between the DNA and the mRNA is that the base Thymine (T) is replaced by the base Uracil (U) in the latter. So, a hypothetical DNA sub-sequence given by *G A C C T A G C C* would produce *C U G G A U C G G* in the mRNA. Note that the genetic code displayed in Table 2.1 is expressed in mRNA codons. The translation process takes place outside the cell nucleus in structures called ribosomes. The mRNA travels from the nucleus to the ribosomes, where its contents are read and the protein is built. There is another kind of RNA, the transfer RNA, which also plays a role in the translation process. Further details can be found in Ridley.[1]

The amino acid is the building block of a protein. Different proteins are built from a different number of amino acids; however, an average number of 100 to 300 amino acids make up a protein. Proteins, on the other hand, account for the features presented by the biological organisms, visible or not: color of the eyes, color of the skin, characteristics of blood cells, among thousands of others. It can roughly be said that a gene is the amount of DNA that encodes for one protein. In the case of the human body, there are around 100,000 genes, and, therefore, approximately 100,000 proteins. More will be seen on the topic of coding DNA in the next chapter of this book.

Finally, before moving on, we must describe a higher level of DNA structural organization, the chromosomal level. DNA is physically carried in structures called chromosomes. In humans, for instance, the DNA is organized in two sets of 23 chromosomes. The existence of a double set of chromosomes defines the important concept of diploidy. These diploid organisms carry a double set of chromosomes, while haploid life forms carry only one set. As a consequence, these organisms carry a double set of genes as well, one set being inherited from the father and the other from the mother. The place where a gene is located on a chromosome is denominated *locus*. For the particular case of diploid organisms, there are two genes occupying each locus. The *genotype* is defined as the combination of the two genes at one locus. The expression of the genotype in the organism is denominated *phenotype*.

Each locus refers to a particular feature of the organism. If an organism presents identical genes at a locus, it is denominated homozygote (concerning a particular feature). If the genes are different, it is called heterozygote. The different forms of the same gene that can be present at a locus are called alleles. It is very common to represent genes by alphabetic letters. For instance, supposing that a particular locus codes for the eyes' color, we could then have, hypothetically, two distinct genes: **A** corresponding to dark eyes; and **a** corresponding to clear eyes. There are two homozygotes' genotypes, **AA** and **aa**, which will produce phenotypes with dark and clear eyes respectively. In the case of the heterozygote genotypes, **Aa**, there are two possibilities: the phenotype can be an intermediate between dark and clear eyes, or one gene can be the dominant. Usually, the upper case letter represents

the dominant gene, while the lower case represents the recessive gene. So, in this particular case, the heterozygote would present dark eyes.

Diploid organisms are more evolved since they are endowed with more robustness to environmental changes.[3] This topic will be further discussed in the next chapter.

2.1.3 Recombination and Mutation

As previously stated, the variation of fitness among individuals of a population is the main fuel of natural selection. This fitness variation is mainly derived from the genetic differences among these individuals. Even if an initial population of individuals has an acceptable degree of variation, natural mechanisms must exist to keep and increase the fitness variation. *Recombination* and *mutation* are such mechanisms.

Recombination is a random process that occurs in sexual populations through interbreeding, producing individuals with new genetic profiles in successive generations. This random process occurs during the event of meiosis, which is responsible for the production of gametes for reproduction. More specifically, the meiosis is a kind of cell division, typical of diploid organisms, when the gametes (haploid cells) are formed from diploid cells. As described in Ridley:[1]

> *At recombination the pair of chromosomes physically line up, and, at certain places, their strands join together and recombine. Recombination shuffles the combinations of genes....*

A graphical visualization of this process is seen in Figure 2.2. In this figure, there are two strands (also called chromatids) that present the following genetic sub-sequences: *A b c D* and *a b c d*. One chromatid was inherited from the father, and the other was inherited from the mother. If the point where the two strands join during recombination (called crossover point) happens to fall between the loci *Cc* and *Dd*, then the following sub-sequences will be produced: *A b c d* and *a b c D*.

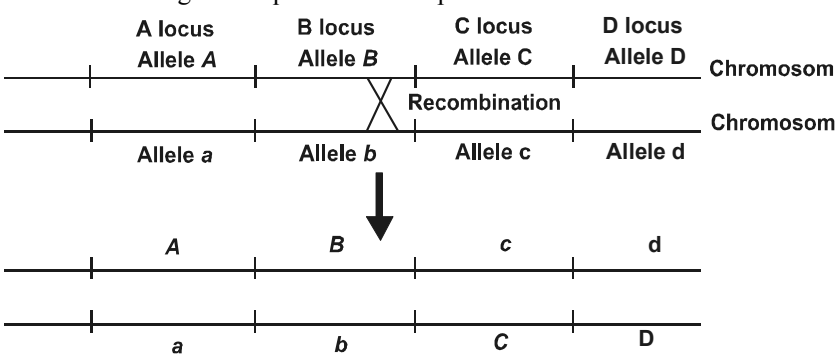


Figure 2.2 Recombination process.

Mutation is another process that is responsible for increasing the genetic diversity.

This process happens in the advent of cell reproduction, where the DNA is physically replicated. Sometimes, due to errors in the DNA replication process, the new cell may have a slightly different DNA sequence compared to the mother cell. Mutational processes will engender diversity in a population if it happens in the production of gametes, since the offspring will present distinct genetic material from their progenitors.

There are many types of mutation observed in biological organisms, and they are usually characterized by the change of one nucleotide into another one. We can divide mutation into *transitions* and *transversions*. The former is characterized by the change of one pyramidine to the other, or one purine to the other (A and G, C and T); in the latter, a purine turns into a pyramidine or vice versa. There is another important distinction between mutations: there are silent mutations and meaningful mutations. The former ones, despite changing a base, do not change the amino acid produced; the latter ones change the amino acid produced by the sequences. The concept of meaningful and silent mutation will be further discussed in this book.

Mutation rates can be experimentally estimated, and a very low value around 10^{-6} per genes has been found for many species, including human beings. Besides these so-called intrinsic mutations, the environment can contribute to increase the rate of mutations; organisms exposed to some chemicals, X-rays, and other factors are likely to have an increased mutation rate.

2.1.4 Mendelian Ratios

The concepts introduced above provide the tools to perform the calculation of the Mendelian ratios, which are the basis of population genetics. When two individuals of given genotypes mate together, the proportions of genotypes in their offspring appear in the so-called Mendelian ratios. Generally speaking, Mendelism is an atomistic theory of heredity, in the sense that not only discrete genes encode for discrete proteins, but the genes are also preserved during development and passed unaltered to the next generation. This theory was opposed to the Blending mechanism, a theory that excluded the concept of gene preservation. The idea that genes are preserved over the generations under Mendelian heredity provided a more powerful framework for natural selection to operate, creating the "Neo-Darwinism" theory.

We will now proceed to some basic calculations of these ratios. Three cases will be briefly described in three examples: i) a mating considering only genes at one locus; ii) a mating considering two loci in different chromosomes; and iii) a mating considering two loci in the same chromosome[1]. Genes that are on the same chromosome are denominated linked, while genes on different chromosomes are unlinked. In the human species, there are around 5000 genes per chromosome (100,000 genes distributed over 23 chromosomes).

i) AA x Aa

In this case, the individual *AA* will have 100% of the gametes with gene *A*. The

individual *Aa* will produce 50% gametes *A* and 50% gametes *a*. Therefore, 50% of the offspring will be *AA*, while the other 50% will be *Aa*.

In another possible case, if two heterozygotes *Aa* mate, we will have 25% AA, 25% aa, and 50% Aa. The latter can be visualized by the graph shown in Figure 2.3.

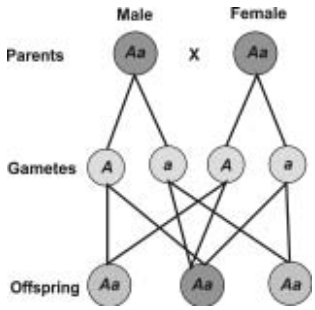


Figure 2.3 Mendelian ratio.

ii) AB/Ab x AB/ab unlinked

In this case, the loci for genes Aa and Bb are on different chromosomes, so they are independently divided or segregated in the meiosis events. The first parent will have the following gametes distribution: 50% A/B and 50% A/b. The second parent, on the other hand, will have the following gametes' distribution: 25% A/B; 25% A/b; 25% a/B; 25% a/b. So, multiplying these statistics:

A 50% A/B x 25% A/B = 12.5% AB/AB
B 50% A/B x 25% A/b = 12.5% AB/Ab
C 50% A/B x 25% a/B = 12.5% AB/aB
D 50% A/B x 25% a/b = 12.5% AB/ab
E 50% A/b x 25% A/B = 12.5% Ab/AB
F 50% A/b x 25% A/b = 12.5% Ab/Ab
G 50% A/b x 25% a/B = 12.5% Ab/aB
H 50% A/b x 25% a/b = 12.5% Ab/ab

Table 2.2 below summarizes the offspring distribution.

Table 2.2 Offspring distribution for the mating between AB/Ab x AB/ab, considering independent segregation

Offspring	Proportion	Case
AB/AB	12.5%	A
AB/Ab	25%	B + E
Ab/Ab	12.5%	F
AB/aB	12.5%	C
AB/ab	12.5%	D + G
Ab/ab	12.5%	H

iii) AB/Ab x AB/ab linked

In this particular case, the two loci are located in the same chromosome. During recombination, the contents of the chromosomes' strands are spliced with a particular probability. If we define the variable *r* as the probability that the crossover point hits a point between loci A and B, the first parent will present the following gametes' distribution:

(2.1)

$$P(AB) = ½ (1 - r) + ½ r = ½$$

$$P(Ab) = ½ (1 - r) + ½ r = ½$$

The probability of having an A/B gamete, P(A/B), is given by the sum of two components: *½ (1– r)*, in case the crossover point does not lie between A and B, and *½ r* in case the crossover point lies between A and B. Either way a gamete A/B will be produced. The same situation applies to the gamete A/b. In the case of the second parent, the following gamete distribution will be observed:

(2.2)

$$P(AB) = ½ (1 - r)$$
$$P(Ab) = ½ r$$
$$P(aB) = ½ r$$
$$P(ab) = ½ (1 - r)$$

In this case, the gametes' distribution depends on the probability *r*. We will then have the following offspring ratios:

(2.3)

$$P(AB/AB) = ½ \cdot ½ (1 - r) = ¼ (1 - r)$$
$$P(AB/Ab) = ½ \cdot ½ \cdot r + ½ \cdot ½ (1 - r) = ¼$$
$$P(AB/aB) = ½ \cdot ½ \cdot r = ¼ \cdot r$$
$$P(AB/ab) = ½ \cdot ½ \cdot (1 - r) = ¼ \cdot (1 - r)$$
$$P(Ab/Ab) = ½ \cdot ½ \cdot r = ¼ \cdot r$$
$$P(Ab/aB) = ½ \cdot ½ \cdot r = ¼ \cdot r$$
$$P(Ab/ab) = ½ \cdot ½ (1 - r) = ¼ (1 - r)$$

It is very important to note that, contrasting to the case of unlinked genes, we can not say, for instance, that AB/ab is the same of Ab/aB: in the former, the genes A and B are in the same strand, and in the latter, they are located on different strands.

2.1.5 The Evidences of Evolution

When it was first presented, the evolutionary theory raised much suspicions, particularly because it was dealing with religious beliefs about the origin of life. Nowadays, the evolutionary hypothesis is widely accepted by the scientific community, mainly because of the existence of many facts that strongly support this hypothesis:

- On a small scale, evolution was already seen to take place in nature, in the case of moths or in artificial selection experiments.

- Similarities between species suggest that they have descended from a common ancestor.

- The fossil record provides direct evidence of the origin of new species and of the transformation of species.

Nonetheless, there is still a large number of open questions surrounding natural evolution that makes this area a very attractive field of research.

Provided that science achieved a partial understanding of the way nature works, we can roughly contrast the by-products of evolution, the biological organisms, with human designed artificial systems:

- Even the simplest life forms are more complex and more difficult to understand than the most complex human designed engineering systems; the mammals' nervous systems and the form of interactions in insects' colonies are examples of complex systems evolved by nature.

- Natural systems are far more robust than their artificial counterparts, and also embedded with interesting properties such as self-repair. Interested readers can consult the work of Thompson[4], where an analogy between temperature compensation mechanisms in natural and artificial systems is drawn.

- Biological organisms seem to be much less parsimonious than artificial systems; for instance, the amount of DNA present in most organisms greatly exceeds the amount needed to encode for proteins; chapter 3 will discuss more on this particular issue.

- Obviously, the time needed for engineers to come up with a design is negligible when compared to the geological time taken by nature to evolve even simple life forms.

Based on the above observations, computer scientists and engineers started to ask if the use of principles employed by nature could bring some benefit to the design of artificial systems. The area of evolutionary computation was created partly as an attempt to shed light into this important question.

2.2 PUTTING THE IDEAS TO WORK: EVOLUTIONARY COMPUTATION

The attempt to model the natural mechanisms to solve problems in general gave birth to Evolutionary Computation. This research field encompasses a class of algorithms that employ some common concepts, which usually bears some similarity with the basic features observed in natural evolution. We first present these common characteristics, providing the basis for the reader to understand the major evolutionary algorithms, whose descriptions follow in the next section.

Since the 40's, computer scientists have attempted to use natural mechanisms as metaphors for computation. Artificial Neural Networks[5] is certainly the most popular example, in which the nervous system served as inspiration to new computational algorithms. At the end of the 50's, scientists began looking at principles of biological evolution to develop new models for computation. Optimization problems have been the main target application of these new computational models. Working independently, some scientists had developed algorithms based on similar principles; afterwards, other researchers compiled the ideas derived from these early efforts, building more powerful algorithms, which are now applied to a wide variety of problems. These tools received the name of *Evolutionary Algorithms*.

Evolutionary Algorithms are typically applied to solve search problems of the form:

$$f : S \rightarrow \Re$$

where S is a search space that is constituted by all the possible solutions to a particular problem. Depending on the particularities of the problem, the solutions may be represented by n-dimensional vectors of binary, integer or real numbers, or more complex structures. To all solutions existent in the S domain, a real number is associated, providing a measure of how suitable a solution is to solve the problem at hand. The primary task of an evolutionary algorithm is to efficiently sample a very large search space S, and find solutions that conform with the objective of the problem. It is important to mention that we are not necessarily talking about *optimal* solutions, but about *satisfactory* solutions. Whenever dealing with large and complex search spaces, optimality can be difficult to achieve and we can only hope to find a satisfactory solution.

In order to contrast the notions of optimality and acceptability of solutions, we can always emphasize the fact that real problems are getting more and more complex nowadays. For instance:

- High-speed, micropower, and low-area silicon chips are increasingly demanded by the electronics industry, particularly in telecommunications and aerospace applications.

- The highly linked global economy that characterizes the world today turns investment decisions into very complex search problems.

- Processing increasingly larger databases, usually made up of millions of registers, and trying to draw conclusions from this kind of historical data turn out to be an unfeasible task without a good search technique.

Therefore, our basic claim is that traditional optimization or analytical tools may not be suited to the complex problems that will arise in the future and, in fact, are already being handled nowadays.

Having established the basic reasons to investigate efficient search tools, we will now introduce one of them, Evolutionary Algorithms. Evolutionary techniques can be roughly defined as interactive algorithms that improve the performance of a population of potential solutions, with respect to a particular search problem, through the application of operators inspired in natural evolution. The operators employed by evolutionary algorithms are inspired in the main mechanisms of natural evolution described in the first part of this chapter: natural selection; recombination or crossover; and mutation.

Given a particular search problem, a suitable representation will have to be selected to encode the potential solutions into data structures of the type defined in the set *S*. After the representation is chosen, a number *N* of potential solutions, also called individuals, is randomly generated. These individuals undergo two basic steps: evaluation and genetic operations. During the evaluation process, a real number, also called fitness, is assigned to each individual. The individual fitness provides a measure of how adequate the individual is to satisfy the particular problem specification. After being evaluated, the following genetic operations may be applied: selection, crossover, and mutation. As their names imply, they are inspired by their natural counterparts. After going through these so-called genetic operators, a new population is then produced, constituting the next generation. The evaluation and genetic operators are applied to the next generation and the process goes on like this, until a stop criterion is met. This stop criterion may be a maximum number of generations or the achievement of a compliant solution to the problem. Figure 2.4 depicts the basic flow of an Evolutionary Algorithm (EA).

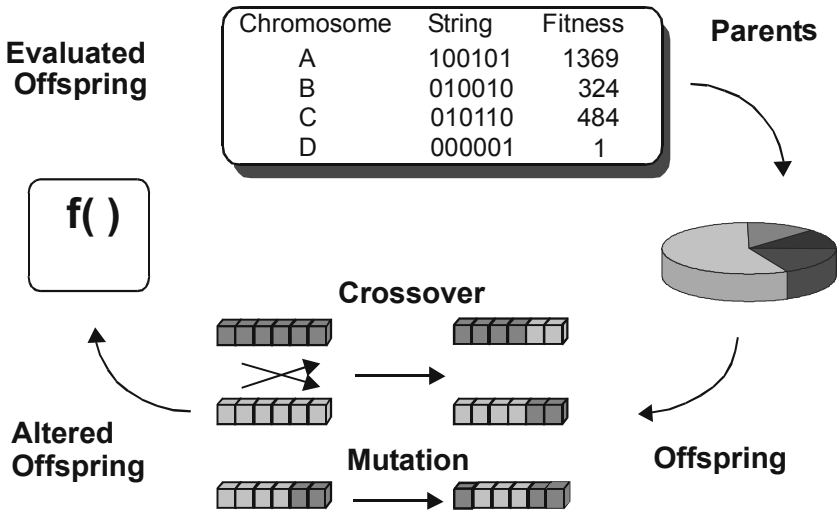


Figure 2.4 Basic flow of an Evolutionary Algorithm.

The diagram of Figure 2.4 sketches somewhat superficially the operation of an evolutionary algorithm. A more detailed description is given next, by analyzing the main components introduced above, i.e., representation, evaluation, and genetic operators.

2.2.1 Representation

The representation refers to the way the solutions to a given problem are encoded into a data structure which can be processed by a digital computer. A solution to a given problem can be:

- A real number, representing the optimum value of a mathematical function
- A vector of real numbers, representing the optimum value of a multi-variable function
- A list of events, which can represent an optimal order of events in the implementation of a particular task
- A structure, which can symbolize some engineering design, such as an electric circuit, a mechanism, a civil engineering structure, a chemical reaction, or any other engineering system

The first two examples mentioned in the list above have a straightforward representation, since real numbers can be easily manipulated with a very high precision by digital computers. The third example exhibits very direct mapping as well – a list of actions or events can be represented by a vector, where each particular component is a pointer to an event or an action. For instance, if one wishes to optimize the cost of a particular manufacturing process, it is usually necessary to find the optimal order in which the actions required to accomplish the process are carried out. A list of events can also represent the solution for search problems in Artificial Intelligence (AI), where strategies need to be conceived. The fourth example illustrates the most interesting case from the point of view of this book: how to encode the solution of an engineering problem into a data structure. An electronic circuit, a chemical reaction, or a particular mechanism, such as a motor, are examples of arbitrary structures. They usually do not present an obvious representation, and it is necessary to work out the means whereby they can be encoded in order to be processed by the evolutionary system.

For representing solutions in these classes of applications, EAs usually employ one of the following data structures:

- Binary strings
- Integer or real-valued vectors
- Finite state representations
- Parse trees

Before describing these four representations, we will define the concept of *cardinality of the representation alphabet*. This concept can be easily understood if we think about the biological case, in which there are four different basies, *A*, *T*, *G* and *C*, present in the DNA encoding. Likewise, artificial chromosomes will use symbols to encode solutions. The number of different representation symbols is the alphabet cardinality. These symbols are usually numeric ones, such as binary, integer, or real values. In the case of a binary representation, the alphabet cardinality is 2, since only two symbols may exist: **0** or **1**. In the case of integer representation, the alphabet cardinality is arbitrary. In the case of real representation, there is an infinite number of symbols in the encoding process. Figure 2.5 illustrates hypothetical string structures using binary, integer, and real symbols respectively. Even though the string arrangement for the representational symbols seems to be the most obvious, other arrangements are possible, such as trees or state machines representation. That is what is now discussed.

1	**1**	**0**	**1**	**0**	**0**	**1**

13	**7**	**24**	**19**	**3**	**47**	**36**

2.589	**7.98**	**0.764**	**1.9**	**13.3**	**1.013**	**6.36**

Figure 2.5 Examples of different representational symbols for linear chromosomes.

Binary strings have been commonly employed by EA practitioners as a way to represent solutions. There exists a mathematical formulation, called *Schema Theorem* (formally defined later in this chapter), which suggests that binary strings generally provide a more efficient representation. However, the main disadvantage of the binary representation is that it is not a natural one for most optimization and search problems, as described in Bäck:[6]

> *At present, there are neither clear theoretical nor empirical arguments that a binary representation should be used for arbitrary problems other than those that have a canonical representation as pseudo-Boolean optimization problems. From an optimization point of view, where the quality of solutions represented by individuals in a population is to be maximized, the interpretation of genetic algorithms as schema processors and the corresponding implicit parallelism and schema theorem results are of no practical use. From our point of view , the decoding function $\Gamma:\{0,1\}^l \rightarrow S$ that maps the binary representation to the canonical search space a problem is defined on plays a much more crucial role than the schema processing aspect, because depending on the properties of Γ, the overall pseudo-Boolean optimization problem $f' = f^o \Gamma$ might become more complex than the original search problem $f: \Gamma \rightarrow \Re$. Consequently, one might propose the requirement that, if a mapping between representation and search space is used at all, it should be kept as simple as possible, and obey some structure preserving conditions that still need to be formulated as a guideline for finding a suitable encoding.*

In the context of Genetic Algorithms (GAs), the schema theorem suggests that binary representation should be beneficial. Bäck argues that, when using a binary representation, a decoding function $\Gamma:\{0,1\}^l \rightarrow S$ actually performs the mapping between the binary world and the real search space *S*. He claims that any theoretical advantage binary representation may confer to Evolutionary Algorithms is certainly counter-balanced by the limitations of the mapping Γ. We will talk more about binary representation later in this chapter, when we present Genetic Algorithms.

The use of integer or real valued strings or vectors usually provides a more

simple and direct representation, being often considered the best choice. Particularly, the evolutionary algorithms called Evolutionary Strategies and Evolutionary Programming (formally defined in the next section) operate on real numbers. The paragraph below, extracted from Fogel,[7] summarizes the advantageous features of this symbolic encoding:

> *When posed with a real-valued function optimization problem of the form find a vector* $\mathbf{x}$ *such that F(x):* $\Re^n \to \Re$ *is minimized (or maximized), evolution strategies*[8] *and evolutionary programming*[9] *typically operate directly on the real-valued vector x (with the components of* $\mathbf{x}$ *identified as object parameters). In contrast, traditional genetic algorithms operate on coding (often binary) of the vector* $\mathbf{x}$*.*[3] *The choice to use a separate coding rather than operating on the parameters themselves relies on the fundamental belief that it is useful to operate on subsections of a problem and try to optimize these subsections (i.e. building blocks) in isolation and then subsequently recombine them so as to generate improved solutions.*

Contrasting with these linear string representations, Finite State Machines and Parse Trees consist of more complex data structures to represent solutions to problems. We can briefly review the concept of a finite state machine, which is defined as a 5-tuple $M = (Q, \tau, \rho, s, o)$.[7] Q is a finite set of states, τ is the set of input symbols, ρ is the set of output symbols, s is the next state function, and o is the next output function. The function s is defined as:

$$s: Q \times \tau \to Q,$$

and o is defined as:

$$o: Q \times \tau \to \rho$$

Figure 2.6 below shows an example of a three-state finite machine.

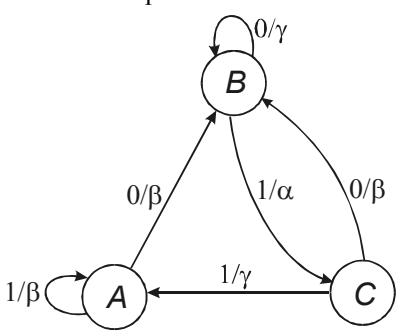


Figure 2.6 Three-state finite machine.

Finite state representations may be particular useful in prediction problems that can be represented as a task to forecast the next symbol in a sequence of symbols. This problem was first suggested by Fogel et al.[10]

Another kind of encoding that does not follow the linear string model is the parse tree representation. Figure 2.7 illustrates an example of such a parse tree representation for a mathematical expression. More will be said about this representation later in this chapter, when we present the Genetic Programming technique.

It is up to the user to choose which data structure will suit better to represent the solution to a given problem, and this choice will have a major impact on the EA performance. There is no set of rules to choose the best representation to a given problem, but we always suggest to try, at first, the representation that seems more obvious in the domain of the particular problem.

We can also distinguish between two representational procedures of EAs, *fixed-length representation* and *variable length representation*. In the former, the chromosome data structure has a fixed size, while, in the latter, we can observe chromosomes of different sizes within a population, and/or between different generations. Considering the four types of representation discussed above, it can be verified that some of them, such as the parse trees, are intrinsically variable in chromosome length. As it will be soon explained, the use of parse tree data structures only makes sense when they can present variable sizes and shapes. In contrast, the use of binary strings, integer or real-valued vectors, and finite state representations may be or not of variable length.

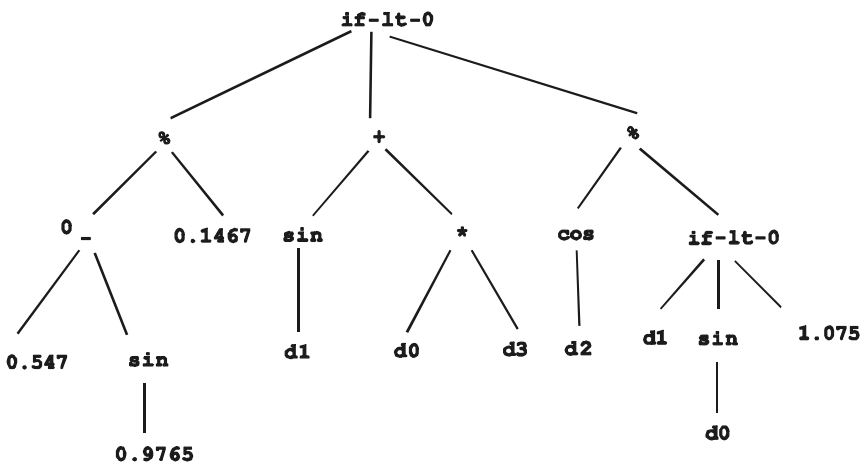


Figure 2.7 An example of parse tree representation for a complex numerical function.

Fixed length representation is more commonly used due to its simplicity. However, it is a limited form of representation and it is far away from the kind of mapping observed in nature. On the other hand, variable length representation is rather more flexible, being able to map structures of different sizes and forms. The disadvantage of variable length systems is that it endows evolutionary algorithms with extra

properties that require the incorporation of new methods into the algorithm; control of the solution size and the inclusion of new operators are some of them. The topic of Variable Length Representation (VLR) systems will be stressed throughout this book.

2.2.2 Evaluation

The evaluation step refers to the process of assigning a fitness value to each chromosome or individual sampled by the evolutionary algorithm. In nature, the fitness of an individual specifies how well adapted he or she is to a particular environment. Similarly, in the case of evolutionary algorithms, fitness provides a measure of how well an individual (a solution) performs according to a particular problem specification. This measure is provided by a scalar value, integer or real. In the case of the standard definition of search problems:

$$f : S \rightarrow \Re$$

We can observe that a real number, the fitness value, is associated with each point in the search space. When there is only one objective to be satisfied in a particular search problem, the fitness evaluation function is usually very straightforward. However, when the satisfaction of more than one objective must be pursued by the Evolutionary Algorithm, finding the fitness evaluation function is a very complex task. This issue is further discussed in the next chapter.

2.2.3 Main Operators

There are three natural mechanisms in which artificial evolutionary algorithms are based on: *natural selection*, *recombination,* and *mutation*. They are the main operators of Evolutionary Algorithms, being called genetic operators. We now discuss how these mechanisms have been computationally formulated.

2.2.3.1 Selection

The selection operator is an essential component of Evolutionary Algorithms[11] . We can find in the literature five implementation approaches for this operator: proportional selection; tournament selection; truncation selection; linear rank selection; and exponential rank selection. Selection mechanisms are characterized by the *selection pressure* or *selection intensity* they introduce in the Evolutionary Algorithm[12] . The term selection pressure is used with different connotations in the context of artificial evolution. In this book, we will adopt the definition employed in genetics which states that selection intensity is the variation on the average population fitness induced by the selection method. Formally, the selection intensity I can be defined as:

$$I = \frac{M^* - M}{\sigma} \tag{2.4}$$

where M is the current average fitness of the population and M^* is the expected average fitness of the population after selection. σ is the standard deviation of the population's fitness values prior to selection.

Let us then introduce the five basic selection methods according to the selection pressure induced by them.

In the case of *proportional selection*, the probability of selecting a particular individual is simply proportional to its fitness:

$$p_i = \frac{f_i}{NM} \tag{2.5}$$

where p_i is the probability that a particular individual i will be selected; f_i is the fitness of this individual; and N is the population size. It can be demonstrated[12] that, in this case, the selection intensity is given by the rate between the standard deviation and average value of the fitness distribution of the population:

$$I = \frac{\sigma}{M} \tag{2.6}$$

Proportional selection has two potential problems: super-individuals and close competition. The former occurs when one individual presents a fitness much higher than the rest of the population, being called super-individual. The super-individual will soon dominate the selection process and cause a premature convergence of the algorithm. Close competition occurs whenever a group of individuals or all individuals present very similar, not necessarily identical, fitness values; in this case the selection pressure will be much lower than the desirable. In order to overcome these problems, other selection methods have been proposed.

Tournament selection is accomplished by selecting the fittest element among a group of t individuals randomly chosen. The selection intensity is given by the following integral equation:

$$I = \int_{-\infty}^{\infty} t \cdot x \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{y^2}{2}} dy \right)^{t-1} dx \tag{2.7}$$

where x and y are integration variables that represent the fitness distribution of the population. The numerical solution of the above equation can be found elsewhere[12]. From this solution, it can be observed that the selection pressure, as expected, increases with the number of individuals t involved in the tournament. For a population of size N, N tournaments will happen to generate N new individuals.

In the case of *truncation selection*, an arbitrary threshold T is imposed to the selection process. The T individuals with highest fitness can be selected with the same probability, while the remaining ones are excluded from the selection

process. It can be demonstrated that the corresponding selection pressure is given by:

$$I = \frac{1}{T} \cdot \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{f_c^2}{2}} \tag{2.8}$$

where f_c is the lowest fitness value among the T best individuals. If we plot a graph of the selection intensity I as a function of the threshold T, we will observe that selection pressure diminishes as T increases, as one would expect.

The *linear rank* method is perhaps the most widely used by the Evolutionary Computation community. The individuals are initially sorted according to their fitness, whose values are changed based on the relative position of each individual. To the best and worst individuals, fitness of η^+ and η^- are respectively assigned. These two values can be arbitrarily determined, but the original formulation states that two conditions should be obeyed: $\eta^+ = 2 - \eta^-$ and $\eta^- \geq 0$. The other individuals will have their new fitness values linearly distributed between η^- and η^+, assigned according to their sorting position. The selection intensity is given by:

$$I = (1 - \eta^-)\frac{1}{\sqrt{\pi}} \tag{2.9}$$

From the above equation we can conclude that the selection pressure reduces as the value of η^- increases.

The *exponential rank selection* method differs from linear rank in that the selection probabilities follow an exponential function. The probability function is given by:

$$p_i = \frac{c-1}{c^N - 1} c^{N-i}; \qquad i \in \{1,\ldots,N\} \tag{2.10}$$

where c, which varies from 0 to 1, determines the exponential level of the method. The closer this value is to 1, the lower is the function's exponential level, and, as a consequence, the selection intensity. The selection intensity is given by the following expression:

$$I \approx \frac{\ln(k)}{-2.548 - 1.086\sqrt{k} + 0.4028\ln(k)} \tag{2.11}$$

where $k=c^N$. If we plot I as a function of c, we will observe that the selection pressure reduces with c.

2.2.3.2 Crossover

As described previously, this operator is inspired in the idea of genetic material recombination between individuals. In natural systems, recombination takes place during the event of meiosis, as explained in section 2.1. According to the EA's basic flow depicted in Figure 2.4, crossover takes place after selection. The crossover

operator is probabilistically applied to the pool of *N* selected individuals. Two individuals from this pool are randomly selected, and, according to a pre-defined probability, their genetic material is spliced or not. If splicing occurs, new individuals with genetic material of both parents are produced; otherwise, the parents, with unchanged genetic material, are passed to the next step of the reproduction process. A new pool of *N* individuals is produced after the process of randomly picking two individuals is repeated *N/2* times.

The probability at which the two individuals will have their genetic material spliced is denominated crossover rate. This rate is usually kept at high levels, above 60%. Nonetheless, for most applications, the sensitivity of the EA performance to this parameter is affected by other features of the EA such as the selection method and the replacement policy of the population.

There are many ways in which two individuals may have their genetic material spliced. Particularly, the literature reports three standard sorts of crossover: one-point; two-points; and uniform. In the case of one-point or simple crossover, one point is randomly chosen, within the boundaries of the string representing the chromosome, as the crossover point; then, two individuals are generated with the segments defined by the crossover point exchanged. In the case of two-point crossover, two cutting points are randomly chosen, and the segments exchanged. In the case of uniform crossover, a random binary pattern will determine the contribution of each parent to their offspring. Figure 2.8 illustrates these three types of crossover.

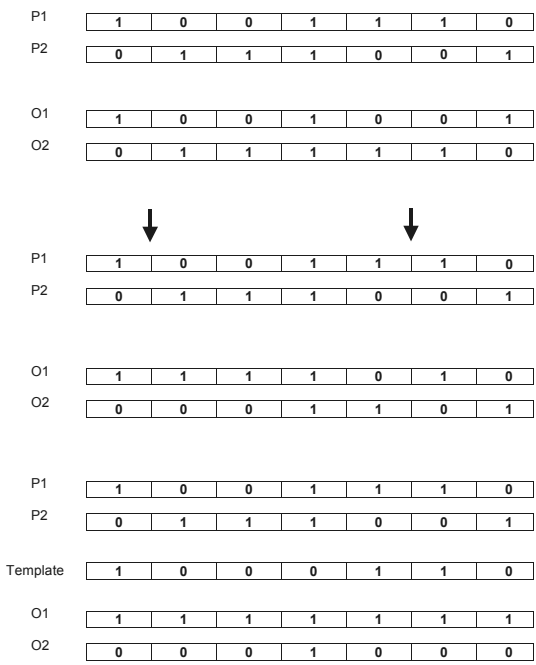


Figure 2.8 Illustration of the three types of crossover: (A) one-point crossover; (B) two-point crossover; (C) uniform crossover.

While one-point and two-point crossover have a local scope, tending to preserve features that are encoded in a compact form in the chromosome, uniform crossover can combine arbitrary patterns present in the chromosomes. As a consequence, uniform crossover is also more likely to destroy good features of the individuals. We remark that there are also reports of new versions of crossover involving more than two individuals. In fact, one can think of many ways of implementing this operator, some times departing from the principles of the recombination processes observed in nature.

2.2.3.3 Mutation

The mutation operator provides an exploratory behavior, in the sense that it induces the Evolutionary Algorithm to sample new points of the search space. If an Evolutionary Algorithm is developed based solely on selection and crossover, it can be observed that the system soon converges. This stems from the fact that crossover is very limited in generating new individuals after some generations. Therefore, the mutation operator is essential to keep the diversity and renew the genetic material.

Mutation is typically the last step of the reproduction process. After crossover, there is a pool of N individuals, some of them different from their parents due to recombination. Mutation is applied at a particular rate to each locus of all the individuals. After mutation, a new pool of N individuals, the new generation, is created. The new generation then undergoes the same processes of evaluation, selection, crossover, and mutation.

The rate at which the mutation is applied is typically low. The EA performance is very sensitive to the value of this operator, but the best value is problem dependent. If the mutation rate is too low, the EA performance degrades; on the other hand, if it is too high, the evolutionary process will approach a random search. Although the best mutation rates are problem dependent, a value between 1% and 5% is usually recommended. When the mutation is actually applied, the new value for the mutated locus is randomly chosen among the different symbols provided by the representation alphabet.

There is a controversy in the EA community regarding the role of crossover and mutation. Some authors argue that crossover is the driving force of Evolutionary Algorithms,[3] while others claim that mutation plays a more important role.[13] As we will have the opportunity to show in the following section, the role of this operator may change according to the particular kind of Evolutionary Algorithm employed.

Another important aspect of Evolutionary Algorithms is the policy adopted to replace the current population by the new generation. This replacement may be total or partial. In the former, all the individuals of the current population are replaced by the new generation; in the latter, only a particular percentage of the population is replaced. In this last technique, the replaced individuals are the ones with lowest fitness in the population. We call this Steady-State reproduction. A more common strategy is the so-called elitism, in which the best chromosome is always copied to the next generation. Both steady-state and elitism improve the Evolutionary

Algorithm performance in most applications, since they prevent good chromosomes from being eliminated from the population in the selection process.

2.3 THE MAIN EVOLUTIONARY ALGORITHMS

This section briefly reviews historical aspects of the field of evolutionary computation. Four classes of evolutionary algorithms are introduced in this section: Evolutionary Programming (EP); Evolutionary Strategies (ES); Genetic Algorithms (GAs); and Genetic Programming (GP).

The first attempts to incorporate natural evolution principles in computer programming probably happened in the 50's. The works of Friedberg[14] and Fraser[15] were pioneering reports of the use of evolutionary processes in computer problem solving. Next, the work of Bremermann[16] described the first attempt to solve numerical optimization problems through simulated evolution. Finally, during that time, Box[17] developed the *evolutionary operation* method, which involved an evolutionary technique for the design and analysis of industrial experiments.

In the 60's, two well-established evolutionary paradigms were devised: Evolutionary Programming and Evolutionary Strategies. These techniques were followed by Genetic Algorithms and Genetic Programming. They are all described in the following sub-sections.

2.3.1 Evolutionary Programming (EP)

This evolutionary algorithm has been conceived by Lawrence J. Fogel.[18] As described in Bäck et al.,[6] he was stimulated by the following ideas:

> *... Fogel considered intelligence to be based on adapting behavior to meet goals in the range of environments. In turn, prediction was viewed as the key ingredient to intelligent behavior and this suggested a series of experiments on the use of simulated evolution of finite-state machines to forecast non-stationary series with respect to arbitrary criteria.*

Evolutionary programming introduced two important ideas borrowed from natural evolution that were used in subsequent evolutionary algorithms: the processing of a *population* of solutions; and the use of the *mutation* operator. Here we review the basic proposal of evolutionary programming, as described in De Jong et al.:[19]

> *A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. ... A function of the payoff for each symbol indicates the fitness of the machine.*
>
> *Offspring machines were created by randomly mutating each parent machine. Each parent produces offspring. There are five possible models of random mutation that naturally result from*

the description of the machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The deletion of a state and change of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or may be fixed a priori. These offspring are then evaluated over the existing environment in the same manner as their parents.

As described above, a fitness value is assigned according to the performance of the finite-state machine performance in the particular task. A set of state-machines is then *deterministically* selected based on their fitness values. Typically, half of the total machines (the best ones) are chosen to become parents of the so-called next generation. These machines are then mutated, according to the mechanism described above, creating then a new generation of state machines, or a new population.

The basic procedure was applied to several problems in prediction, identification, and automatic control. Recently, new EP procedures have been successfully applied to other problems, such as the travelling salesman problem and continuous function optimization.

2.3.2 Evolutionary Strategies (ES)

Evolutionary strategies were devised in 1965 by Rechenberg and Schwefel. They were initially studying problems in the area of robotics. An algorithm that processed one individual along many generations had initially been devised. A mutation operator applied with binomial distribution was also employed. Some demonstrations of this relatively simple evolutionary algorithm have been performed: in 1965, a student (Lichtfuβ[20]) optimized the shape of a bent pipe. In another experiment, Schwefel[21] applied this method to a well-known problem in the area of mechanical engineering: the two-phase nozzle optimization. Despite the fact that a simple evolutionary strategy with only one individual was used in this experiment, advanced concepts of gene deletion and duplication have been incorporated into the evolutionary algorithm.

In Bäck et al.,[6] the basic scheme of ES is described:

An individual $\boldsymbol{a}$ *consisting of an element* $\boldsymbol{X} \in \Re^n$ *is mutated by adding a normally distributed random vector* $\boldsymbol{Z} \sim N(0, I_n)$ *that is multiplied by a scalar* $s > 0$ ($\boldsymbol{I}_n$ *denotes the unit matrix with rank n). The new point is accepted if it is better than or equal to the old one, otherwise the old point passes to the next iteration. The selection decision is based on a simple comparison of the objective function values of the old and the new point. Assuming*

that the objective function f: $\Re^n \to \Re$ is to be minimized, the simple ES, starting at some point $\mathbf{X}_o \in \Re^n$ is determined by the following iterative scheme:

$$X_{t+1} = \begin{cases} X_t + \sigma_t \cdot Z_t & \text{if} \quad f(X_t + \sigma_t \cdot Z_t) \le f(X_t) \\ X_t & \text{Otherwise} \end{cases} \tag{2.12}$$

where $t \in \mathbf{N}_o$ denotes the iteration counter and where $(Z_t: t \ge 0)$ is a sequence of independent and identically distributed standard random number vectors.

Nonetheless, the so-called (μ+λ) Evolutionary Strategy has achieved more popularity. In this new scheme, also called multi-membered evolutionary strategies, a total of μ individuals makes up the population. λ new individuals are then selected among the original m individuals, with a probability of λ/μ. The new pool of (μ+λ) individuals undergoes a process of mutation, as in the original algorithm, and also a process of recombination, a new and important feature incorporated. In the recombination process, two individuals have their contents spliced in a multi-point crossover.

Besides the use of a population of solutions, there are also other differences between recent and older versions of Evolutionary Strategies. In contemporary ES, an individual consists of an element $x \in \Re^n$ of the search space, plus several individual parameters controlling the individual mutation distribution.[6] For instance, each genome can be represented by the tuple $(\mathbf{X}, \sigma) \in \Re^n \times \Re_+$ that undergoes the genetic operators. σ is a control parameter for the mutation operator, which is now accomplished in two steps:

$$\sigma_{t+1} = \sigma_t \exp(\tau \cdot Z_\tau) \tag{2.13}$$

$$X_{t+1} = X_t + \sigma_{t+1} \cdot Z \tag{2.14}$$

where Z_τ is a standard normal variable, Z is a standard normal random vector, and t is a control parameter.

2.3.3 Genetic Algorithms (GAs)

Genetic Algorithms (GAs) were conceived by John Holland and fellow workers from the University of Michigan.[22] Their objectives were the investigation of adaptive processes in natural systems and the creation of computer programs that displayed similar features with respect to the natural systems investigated. The research aimed to achieve artificial systems presenting certain interesting properties of natu-

ral systems, such as reproduction and self-repair; such features would eliminate the high cost associated with system re-design.

GAs became the most popular evolutionary algorithm due to its successful application in optimization and search problems, particularly in those problems in which the size or complexity of the search space renders infeasible the use of other optimization techniques. Contrasting with other optimization techniques, GAs are blind, in the sense that extra information of the search space, such as the gradient, is not used by these algorithms.

GAs inherited some basic features of the evolutionary algorithms previously defined: the notion of sampling a population of solutions, the use of the mutation and recombination operators, and the probabilistic nature of these operators. However, two new features have been incorporated into GAs as compared with other algorithms:

- The selection of individuals to create a new population is now probabilistic, rather than deterministic.

- Each solution is now represented as a binary string, exploring an analogy with the representation used in digital computers.

Generally, the widely used three-operator genetic algorithm can be explained through the following flow:

- A population of *n* binary strings is randomly created; these binary strings are also called chromosomes, individuals, or genotypes.

- Each individual is evaluated with respect to a particular specification, which is usually the function to be optimized; a scalar value, called fitness, is then assigned to each genotype.

- Based on this scalar value, the population undergoes the process of selection; a set of *n* individuals is probabilistically chosen, in such way that a higher fitness confers higher chances of selection to an individual. Therefore, even though probabilistic, the selection process is biased towards fitter individuals.

- After selection, pairs of individuals are randomly chosen from the selected pool, and with a particular probability, undergo the recombination process, also called crossover. The crossover operator, similar to the case of evolutionary strategies, splices the contents of each pair of individuals, creating two offspring mixing the genetic contents present in their parents' strings. In case the two parents do not undergo the crossover operation, they are copied unchanged to the new pool.

- The mutation operator is applied to the new pool of individuals produced after the application of crossover. Mutation is applied typically with a very low rate to all the bit positions that constitute the chromosome. If successful, it flips the value of the particular bit; otherwise, the bit is left unchanged.

- Finally, after mutation, a new generation of individuals is produced. This new generation goes through the process described above, from the evaluation to the mutation step.

- This cycle repeats until a stop criterion is met, such as a maximum number of generations is reached or a desired solution is found.

Genetic Algorithms were initially intended to process binary strings of fixed length,[3] where each chromosome position or locus is represented by a bit, and the number of loci is kept constant along the evolutionary process. Although the first GAs employed binary representation, an increasing number of applications using integer representation has been reported.[23-24] In this case, each chromosome position can assume K different values, instead of only *0* and *1*. The variable K is also denominated the size of the representation alphabet. Therefore, the alphabet and the chromosome size L determine the size of the search space, which is given by K^L.[25]

We will adopt this broader definition of GAs, which will cover not only binary representation of fixed length, but also integer or real representation, of fixed or variable length. In the case of variable length representation, the chromosomes' sizes will vary along the generations and within the population. This topic will be stressed in the next chapter.

Depending on the problem, we can also define a chromosome as a collection of genes, where each gene is constituted by one or more chromosome's positions. If we define the number of loci in a gene by the variable r and the number of genes in the chromosome by g, then the chromosome size $L = g \cdot r$.

GAs are highly non-linear search algorithms, making it difficult to predict its behavior when varying its parameters. Adjusting GAs can be regarded as a form of art more than a science. There have been attempts to draw a precise mathematical model of Genetic Algorithms; the *Schema Theory*, conceived by John Holland,[22] is the most accepted model. This model is not able to explain many aspects of the complex behavior displayed by GAs, but it helps to understand the way GAs work.

A schema is a particular pattern describing a set of chromosomes in the search space. These chromosomes will have identical symbols in some of their positions or loci. In order to represent a schema, one extra symbol is introduced, the *don't care*, denoted by *****. One chromosome is said to belong to a schema if, for each string position, either the chromosome symbol is equal to the schema, or the schema symbol is don't care. For a binary alphabet (K = 2), there will be three possible symbols for the schemata, **0**, **1**, and *****. For instance, if the chromosomes have three

positions (L = 3), then the chromosomes (1 1 0) and (1 1 1) belong to the schema H = (1 1 *), but (0 1 1) does not. Generally speaking, each individual belongs to a total 2^L different schemata.

We have already discussed that, in a general case, the search space sampled by the GA is given by K^L. Then, the total number of schemata is given by $(K + 1)^L$. For L = 3, there will be $(2 + 1)^3$ or 27 schemata.

Based on the schemata concept, an important result is formulated. Before, though, we need to define the notions of order and length of a schema. The order of a schema *H*, represented by *O(H),* accounts for the number of specific positions (different from *) present in the schema. For instance, in the case of binary representation:

$$H = 0\,1\,1\,*\,1\,*\,* \Rightarrow O(H) = 4$$
$$H = 0\,*\,*\,*\,*\,*\,* \Rightarrow O(H) = 1$$

The schema length, ρ(H), is defined as the distance between its first and last specific position. For instance, in the case of binary representation:

$$H = 0\,1\,1\,*\,1\,*\,* \Rightarrow \rho(H) = 4$$
$$H = 0\,*\,*\,*\,*\,*\,* \Rightarrow \rho(H) = 0$$

Having defined these concepts, we can introduce the main result derived from the schema theory, also called fundamental theorem of GAs. This theorem tries to address the question: given that a particular schema is present in the GA population in a particular generation, will it tend to proliferate or to disappear in the next generations? Here follows the result achieved by Holland:

$$m(H,t+1) \geq m(H,t).\frac{f(H)}{f_m}.\left[1 - p_c.\frac{\rho(H)}{L-1}\right].\left[1 - O(H).p_m\right] \quad (2.15)$$

where:

m(H,t) = number of individuals belonging to the schema *H* at the particular instant or generation *t*

f(H) = average fitness of the schema *H*

f_m = average fitness of the population

p_c = crossover rate

p_m = mutation rate

The above equation states that the number of individuals belonging to the schema *H* at instant *t+1* will depend of four factors. The first one is simply the number of instances of *H* at the current generation *t*. The second factor accounts for the probability the schema *H* will pass through the *selection* step. This probability is given by the ratio between the schema fitness *f(H)* and the population average fitness, f_m. Naturally, fitter schemata will have more chances to survive. The third

factor accounts for the crossover: the schema H will pass unchanged to the next generation if it is not disrupted by crossover, i.e., if it is not broken between the first and last specific positions. Observing the equation above, this probability depends on the crossover rate, p_c, and on the ratio between the schema length and the total chromosome size. Finally, the fourth term accounts for the chance by which the schema H will pass unchanged through the mutation operator. Naturally, this will be a function of the mutation rate and of the schema order (mutations applied to a non-specific position will not alter or destroy the schema). Therefore, the following result derives from this equation: *Short and low-order schemata tend to proliferate or disappear exponentially in the next generations, according to their average fitness*.

Although this is one of the most important mathematical results in GA theory, it is still very limited to explain the complex behavior of GAs, due to the following factors:

- It takes only into account the destructive behavior of the three GA operators, selection, crossover, and mutation. But, what about the role of these operators in building good schemata?

- This theorem suggests that we should use representations where good chromosomes belong to short and low-order schemata. This guideline is usually not followed by GA users, because it is often difficult to have a knowledge about how fit a schema will be before an extensive study of the particular representation is performed. More importantly, there are other effects of the genetic operators, as stated in the above item, that may hide the beneficial effects of this suggested compact representation.

Finally, another consequence of the schema theorem refers to the alphabet cardinality. Suppose that we need to sample a search space of 16 solutions; one could encode solutions using a binary alphabet or, on the extreme side, an alphabet of 16 symbols. In the former case, a chromosome length of four positions (L = 4) will be needed, while in the latter only one position will be necessary. When using binary representation, the total number of processed schemata is given by:

$$(2+1)^4 = 81$$

while in the case of hexadecimal representation, the number of processed schemata is given by:

$$(16+1)^1 = 17$$

Therefore, the binary representation offers a maximum number of schemata, compared to the other cardinalities. This is, in principle, advantageous, since it increases the so-called *implicit parallelism* of GAs: the GA does not only explicitly sample a population of solutions, but it also processes, implicitly, a number of

schemata, which is maximum for the binary representation. This result suggests that using the binary representation will always improve the GA performance. Nevertheless, the superiority of the binary representation has not been proved experimentally, and it is a better strategy to choose an alphabet that fits better into the domain of the particular problem. Generally speaking, there is no reason not to choose an alphabet that will lead to the simplest representation.

2.3.4 Genetic Programming (GP)

Genetic programming was conceived in the early nineties by John Koza, from Stanford University. His basic motivation was the idea of using evolutionary algorithms to evolve computer programs. He identified a very straightforward way of representing computer programs in the context of LISP programming language, whose programs can be easily viewed as a tree structure. Therefore, instead of using binary strings to represent solutions, trees have been used as chromosomes in Genetic Programming.

The GP trees are constituted of nodes of two distinct natures, function nodes or terminal nodes. Function nodes are external vertices of the trees, and usually correspond to the following types of functions:

- Arithmetic operations (+ , – , * , \ , %, etc.)
- Mathematical functions (sin, cos, tan, log, etc.)
- Boolean functions (and, or, not, etc.)
- Conditional operators (if ... then ... else)
- Operators that cause iteration (do ... while)
- Any other function of the problem domain.

On the other hand, terminals are the external nodes, corresponding either to a variable relevant to the problem domain or to a constant. For instance, the expression $x^2 + y$ can be described by the tree illustrated in Figure 2.9.

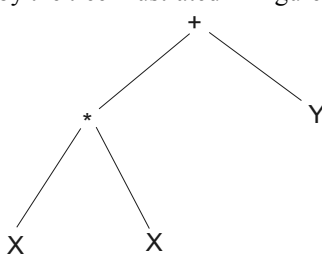


Figure 2.9 Hypothetical example of a tree mapping the expression $x^2 + y$.

Except for this new kind of representation, the basic flow of genetic programming is very much like the one of genetic algorithms. The use of GP to solve a problem encompasses the following steps:

- Determining the terminal set
- Determining the function set
- Determining the fitness function
- Establishing the execution control parameters
- Defining a method to determine the result and finish the execution

These steps are similar to the ones involved in running a GA, except for the fact that a function and a terminal set must be defined. Figure 2.10 roughly depicts the GP cycle, where we can observe that the GA strings are replaced by the GP trees.

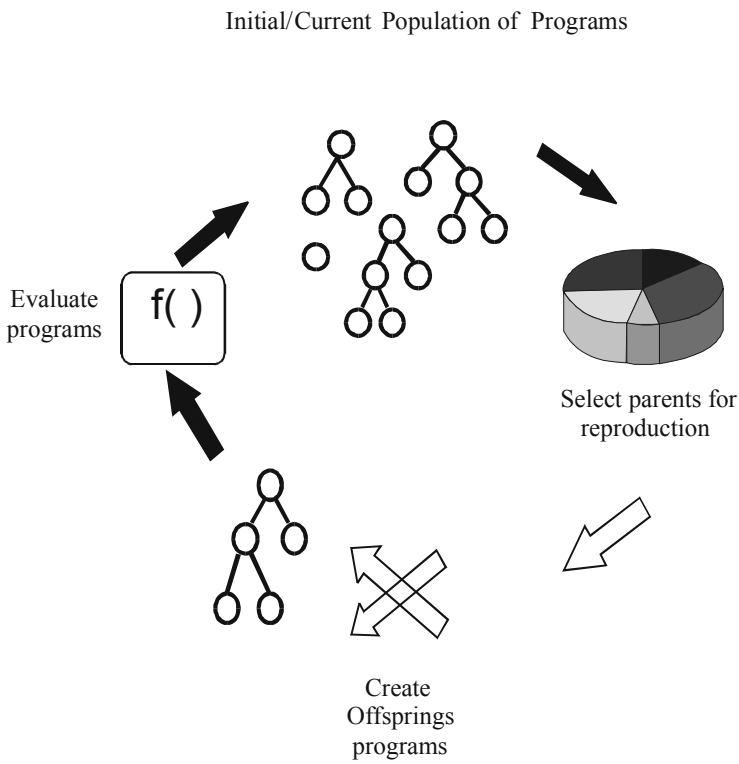


Figure 2.10 Genetic programming cycle.

Let us now be a little more specific about the GP basic flow:

Population initialization: The initial trees are randomly generated by picking up, at random, terminals and functions from the respective sets. There is a parameter controlling the maximum height of the initial trees.

Fitness measure: As in the case of GAs, a scalar value is assigned to each tree, reflecting how well the tree performs according to a particular specification.

Selection: As in GAs, it is probabilistic, favoring the fittest individuals.

Operators: Besides crossover and mutation, there are other possible operators in GP, such as permutation, edition, deletion, encapsulation, and destruction. These operators are described in detail in Koza;[27] in this section, we will only describe the crossover, since this is the most important GP operator.

Termination criteria: When a maximum number of generations or a desired solution is reached.

As mentioned above, the most important operator of GP is the crossover. The crossover role in GP is more important than in GAs. In the case of GAs, crossover promotes a local exploration, while the mutation operator promotes exploitation, stimulating the search in different regions of the search space, and avoiding quick convergence to local optima. Therefore, in GAs, the crossover operator would not be effective without mutation. This is not the case in GPs due to the different chromosome structure. Figure 2.11, Figure 2.12, and Figure 2.13 illustrate the standard crossover operation performed in GP, for the particular case of the evolution of a mathematical expression.

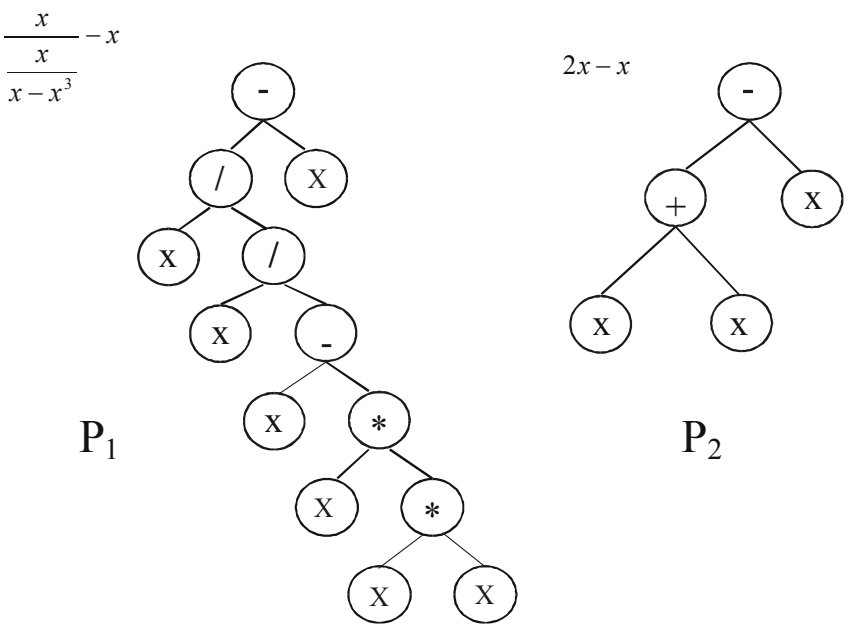


Figure 2.11 First step of GP crossover: choosing parents.

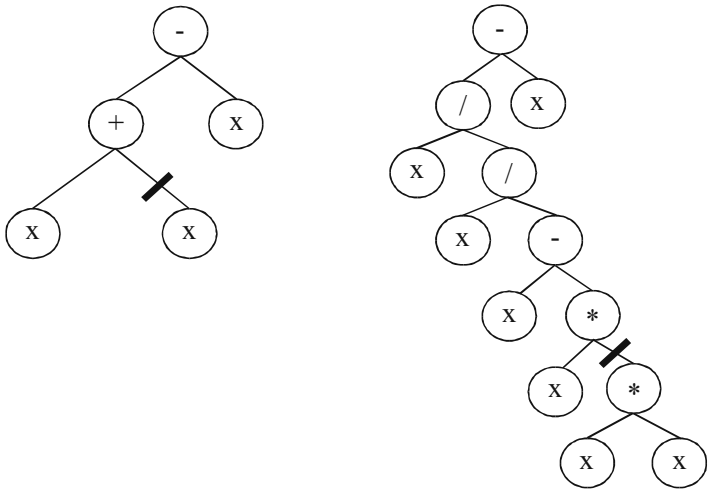


Figure 2.12 Second step of GP crossover: randomly selecting cutting points.

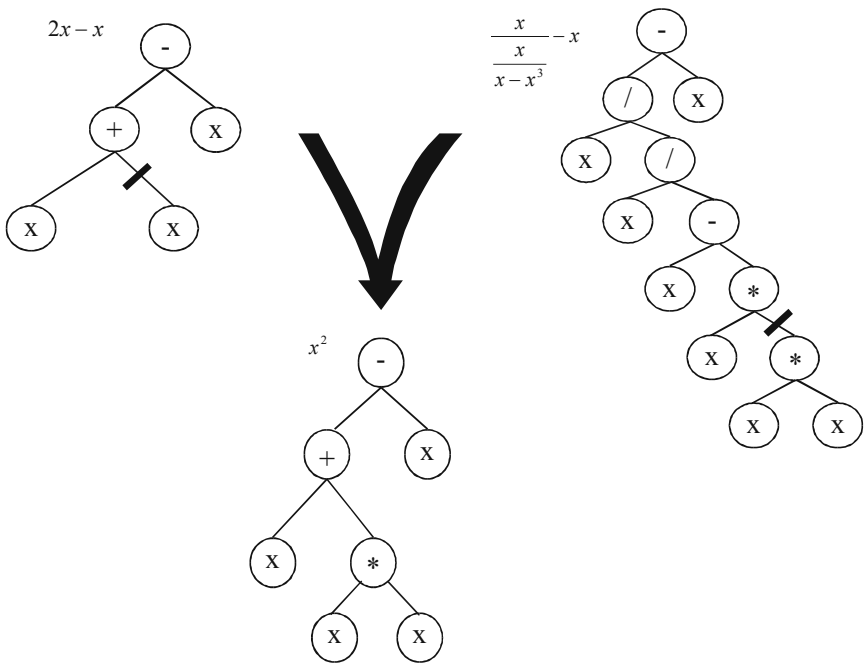


Figure 2.13 Third step of the GP crossover: exchanging sub-trees.

As demonstrated in the figures above, there are basically three steps involved in the application of crossover: choosing two parents from the population; selecting cutting points in the parents' trees; and exchanging the sub-trees to produce the

offspring. There is an important difference between GP and GA regarding the result of applying the crossover operator. In GA, the crossover between two identical parents will produce offspring identical to their parents, regardless of the crossover point. On the other hand, when two identical individuals recombine in GP, the offspring will be identical to their parents only if the selected cutting point is the same in both parents, which is not a probable event. Therefore, crossover in GP will introduce more diversity than in GAs.

One important development in Genetic Programming refers to the use of Automatically Defined Functions (ADFs).[26] As the name implies, ADF has been a way devised to borrow the advantages of using software routines and functions in the program flow. The function set will now include a number of functions defined in the own tree representing the individual. Figure 2.14 depicts the GP tree structure when using one ADF.

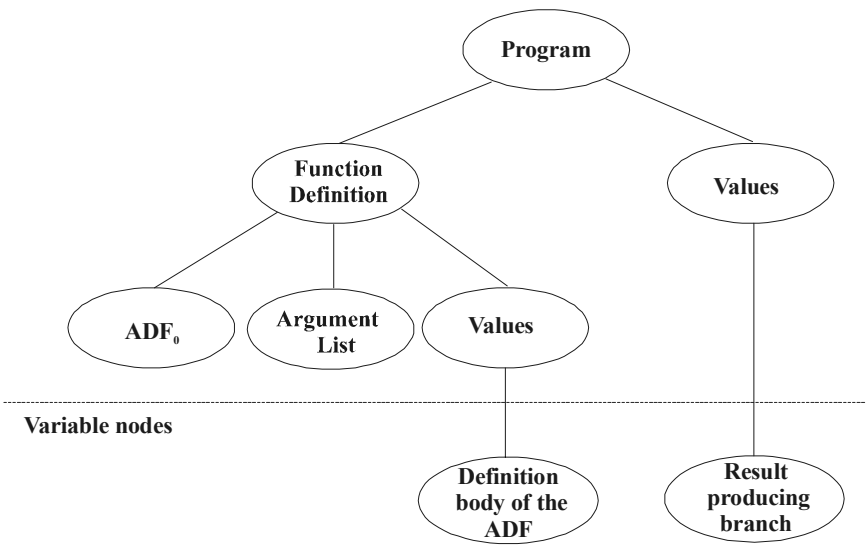


Figure 2.14 Tree structure typical of GP when using Automatically Defined Functions (ADFs).

In the above figure, it can be observed that the tree is divided into two sub-trees. The one on the left, *defun*, defines a function called ADF_0, which operates on three arguments, arg_0, arg_1, and arg_2. The structure on the right is actually decoded to determine the individual's fitness. In this hypothetical example, this last structure calls the ADF defined on the left. It is important to remark that both structures (below the trace line) are evolved by GP; thereby different individuals will have different definitions for ADF_0. In his second book, Koza[26] ran many experiments to evaluate the benefits of using ADFs, and concluded that this approach improves GP performance in many cases.

We will now briefly describe an example of a typical GP application in the

problem of symbolic regression. Symbolic regression consists of finding a mathematical expression that better adjusts the data sample. For instance, we can consider the problem of finding a relationship between diameter and radius, and the period of a planet's orbit around the sun, suggested in Koza.[27] This problem consists of rediscovering Kepler's third law, which state that the orbit period of a planet is proportional to the square root of the orbit's radius to the power of three. Suppose that the following data, referring to the orbit of some planets, are furnished:

Table 2.3 Data relating radius, diameter, and period of the orbit of some planets of the solar system.

Radius	Diameter	Period
0.387	0.384	0.24
0.723	0.972	0.62
1.000	1.000	1.00
1.524	0.533	1.88
5.203	9.775	11.86
9.569	9.469	29.46
19.309	3.692	84.01
30.284	3.492	164.79
39.781	0.447	247.69

In order to tackle this problem, we need first to define the function set. As the problem consists of finding a mathematical expression, the following functions are related to the problem domain:

sin, ln, exp, not , +, –, >, <, if ...then...else

Some of these functions take one argument, others two arguments, and the last one, ***if ...then...else***, takes three arguments.

The terminal set consists of the radius and diameter variables, and the set of real numbers.

The individual's fitness is evaluated by summing, for the eight fitness cases, the absolute deviation between the period produced by the expression and the actual planet period. Figure 2.15 depicts a solution found by GP, which is the correct expression.

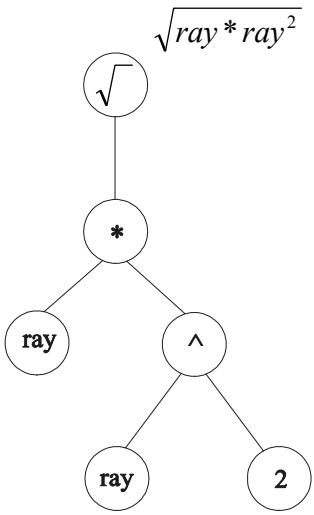


Figure 2.15 100% correct solution found by GP to the problem of re-discovering Kepler's third law.

REFERENCES

[1] Ridley, M., *Evolution*, 2nd ed., Blackwell Science, Cambridge, 1996.

[2] Darwin, C.R., *On the Origin of Species*, John Murray, London, 1859.

[3] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MAs, 1989.

[4] Thompson, A., Temperature in natural and artificial systems, in *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, Husbands, P. and Harvey, I., Eds., MIT Press, Cambridge, 1997, 388.

[5] Churchland, P.S. and Sejnowski, T.J., *The Computational Brain*, MIT Press, Cambridge, 1992.

[6] Bäck,T., Evolutionary algorithms and their standard instances-introduction, in *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, New York, 1997, B1.1.

[7] Bäck,T., Evolutionary algorithms and their standard instances-introduction, in *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, New York, 1997, C1.3:1.

[8] Fogel,D.B., Evolutionary computation models-representation, in *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, New York, 1997, C1.3:1.

[9] Fogel, D.B., *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE, Piscataway, NJ.

[10] Fogel, L.J., Owens, A.J., and Walsh, M.J., *Artificial Intelligence Through Simulated Evolution*, John Wiley & Sons, New York, 1966.

[11] Hancock, P.J.B., An empirical comparison of selection methods in evolutionary algorithms, Evolutionary Computing, AISB Workshop, Leeds, GB, in *Lecture Notes in Computer Science*, 865, Fogarty, T.C., Ed., 1994, 80.

[12] Blickle, T., Theory of Evolutionary Algorithms and Application to System Synthesis, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[13] Harvey, I., The Artificial Evolution of Adaptive Behaviour, Ph.D. thesis, University of Sussex, School of Cognitive and Computing Sciences (COGS), 1993.

[14] Friedberg, R.M., A learning machine: part I , *IBM J*. 2 2-13, 1958.

[15] Fraser, A.S., Simulation of genetic systems by automatic digital computers, *Aust. J. Biol. Sci*., 10, 484, 1957.

[16] Bremermann, H.J., Optimization through evolution and recombination. Self-organizing systems, Yovits, M.C. et al., Eds., Washington, DC, 1962.

[17] Box, G.E.P., Evolutionary operation: a method for increasing industrial productivity, *Appl. Stat*., 6, 81, 1957.

[18] Fogel, L.J., Autonomous automata, *Industrial Res*. 4, 14-9, 1962.

[19] De Jong, K., Fogel D.B., and Schwefel H-P., A history of evolutionary computation, in *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press, New York, 1997, A2.3.

[20] Lichtfuβ, H.J., Evolution eines Rohrkrümmers Dipl.-Ing. thesis, Technical University of Berlin, Hermann Föttinger Institute for Hydrodynamics, Berlin, 1965.

[21] Schwefel, H.P., Experimentelle optimierung einer zweiphasendüse teil I, AEG Research Institute Project MHD-Staustrahlrohr 11034/68, Technical Report 35, 1968.

[22] Holland, J.H., *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.

[23] Grimbleby, J.B., Automatic analogue network synthesis using genetic algorithms, in *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems (GALESIAS - 95)*, England, 1995, 53.

[24] Miller, J.F., Thomson, P., and Fogarty, T., Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: A case study, in *Genetic Algorithms: Recent Advancements and Industrial Applications*, Quagliarella, D. et al., Eds., John Wiley & Sons, New York, 1997.

[25] Pacheco, M.A., Algoritmos Genéticos: Princípios e Aplicações, INTERCON99: V Congreso Internacional de Ingeniería Electrónica, Elétrica Y Sistemas, Lima, 1999, 11.

[26] Koza, J.R., *Genetic Programming II*, MIT Press, Cambridge, Massachusetts, 1994.

[27] Koza, J.R., *Genetic Programming*, MIT Press, Cambridge, 1992.

CHAPTER 3

Advanced Topics in Evolutionary Computation

This chapter extends the previous one by presenting advanced topics in Evolutionary Computation. We select the following issues to be examined: Variable Length Representation (VLR) systems; Evolutionary Algorithms using memory paradigms; Multi-objective Optimization; Speciation in Evolutionary Algorithms; Deception and Epistasy. These new paradigms endow Evolutionary Algorithms with properties that make them more plausible when compared to natural systems. They also turn Evolutionary Algorithms into more efficient tools to solve more complex search problems.

The standard evolutionary computation techniques described in Chapter 2 may have a limited performance when applied to some real world problems. The following features of these problems act as obstacles to the successful application of any particular search technique:

- Real world search problems often require the satisfaction of many objectives.
- They usually have a vast search space, which can not be efficiently sampled, even by evolutionary algorithms.
- The size of the solution, as well as the solution itself, may be unknown to the user.
- Generally speaking, the complexity of the search space, in addition to its size, may be a drawback to solving a particular problem.

This chapter proposes new techniques to handle the class of problems listed above, and contrasts these techniques with the standard methods reported in the literature. This chapter is composed of five sections: the first section analyzes variable length representation systems; the second one introduces the use of memory paradigm in evolutionary algorithms; the third reviews multiple-objective optimization and presents the techniques developed to cope with it; section four examines the use of speciation techniques in evolutionary computation; and the last section discusses the classical concepts of deception and epistasy.

3.1 VARIABLE LENGTH REPRESENTATION SYSTEMS

Variable length representation systems employ genotypes whose effective length varies throughout the evolutionary process and also within the population. The notion of effective length of a genotype should be made clear: it is not, in principle,

the length of the data structure used in the representation, but the fraction of the data structure that is used in the decoding process to construct the phenotype.

There is a twofold motivation for the research in variable length representation systems:

- Drawing a more precise analogy with representation in natural systems
- Developing flexible evolutionary systems that can sample solutions of different sizes

We start this section by substantiating the first argument: variable length representations in natural and artificial systems are described and contrasted. We conclude this section by presenting a new model of variable length evolutionary algorithm, showing details of its performance when applied to some problems in .

3.1.1 Representation in Natural Systems

Let us initially discuss the first motivation for investigating Variable Length Representation (VLR) Evolutionary Algorithms. The second chapter of this book summarized the mechanism of DNA encoding to build an organism. We now go a little deeper into the description of this mechanism. The DNA of organisms in many species is divided into two categories: coding and non-coding DNA. The former effectively contributes to build an organism, because it codes for genes; the latter, in contrast, does not code for genes. We will call these two categories active and inactive DNA, respectively. The role of inactive DNA is a current topic of investigation in biological sciences. The belief that this kind of DNA does not contribute in any way to the well-being of the organism has established the selfish DNA hypothesis.[1] However, some researchers investigate the possibility that inactive DNA may have some structural or regulatory function in the organism. Nevertheless, it is also not clear if inactive DNA can become active, or vice versa, in a particular moment of the organism's life.

A more quantitative estimate of the proportion of active and inactive DNA in human beings can be provided by the following calculation. Human beings use around 100,000 different kinds of proteins throughout their lives. If we assume that there is a one-to-one correspondence between gene and protein (a reasonable approximation), then we can say that there are about 100,000 genes in each human cell. As each protein is constituted by an average of 300 amino acids, and each amino acid is made up of 3 nucleotides, then we can say that one protein is made up of around 10^3 nucleotides. Therefore, one can estimate a total of 10^8 coding nucleotides in the human genome. Nevertheless, it is known that the human DNA is constituted by about 10^9 nucleotides, much more than necessary to code for proteins. This is a very rough estimate, but carefully conducted experiments point to a maximum of 50% of the human DNA as the proportion that may code for proteins. The percentage of inactive DNA varies for different species. For instance,

bacterial DNA seems to be more efficient, since nearly 90% of its DNA is active. Conversely, only around 10% of a salamander DNA is active. The reader can refer to Ridley[1] (1996) for more details.

The origin of inactive DNA is a current topic of research. It is generally believed that they have been accumulated through the generations by the occurrence of failures in basic DNA operations, such as replication and crossing over. They may increase freely in quantity, until the point they may affect the basic functions of a cell, a case in which natural selection starts to check excessive inactive DNA.

Therefore, we may feel tempted to answer the following questions: Would it be interesting to create a simplified model of the complex genotype-phenotype mapping observed in natural system? What kind of benefits, if any, would it bring to evolutionary algorithms? The remaining sections analyze these questions.

3.1.2 Representation in Artificial Systems

In order to discuss the issue of representation in artificial systems, we can contrast standard Genetic Algorithms with Genetic Programming: while the former uses a fixed length representation, the latter uses variable length representation. The strings processed by conventional Genetic Algorithms keep a constant number of bits, each of which contribute to the phenotype construction. On the other hand, GP trees have arbitrary size and shape. Each node contributes to the phenotype, but their number and the way in which they are organized vary for different individuals.

This feature endows GP with more flexibility to represent solutions. If we focus, for instance, on the representation of electronic circuits, this is the natural approach. Whenever an engineer starts to work in a design, he or she does not have a precise idea of the final size of the project. Analogously, in order to tailor evolutionary computation to this class of problems, one must consider these provisions. GP deserves merit for providing this kind of flexible mapping, but this comes at a particular cost: the excessive increase in the tree size along the evolutionary process, known as the bloating problem. Whenever you use GP in a particular problem, a growth in the trees' size along the generations is observed, and they are usually accompanied by a deterioration in the algorithm performance. Control mechanisms must therefore be used to overcome this drawback.[2]

An additional and related problem is the fact that the solutions achieved by GP are usually much bigger in size than the optimal solution to a particular task. Methodologies have been developed to cope with this problem as well, and two examples can be found in the literature: Automatically Defined Functions or ADFs;[3] and Adaptive Parsimonious Pressure.

There are other examples of VLR Evolutionary Algorithms: the Messy GAs;[4] LS-1 classifiers;[5] and the Species Adaptation Genetic Algorithm (SAGA).[6] A survey of these algorithms may be found in Harvey,[6] 1993.

The following section presents a new kind of VLR representation based on variable length strings, conceived by the authors. This representation addresses the problem of achieving a more realistic mapping from the biological point of view, while keeping the technique feasible to handle practical problems. We test this new

approach in a problem in the domain of electronic engineering, the one of passive filters design. Since our VLR representation is based on a string data structure, we can say that this is a new Genetic Algorithm model.

3.1.3 Genetic Algorithms with Variable Length Representation

Genetic Algorithms, in their standard form, process binary strings of fixed length. However, the constraint of binary representation can be relaxed, and integer or real representation can also be used in the context of GAs, as described in the previous chapter. We are now going to relax the fixed length representation constraint of GAs. A new form of chromosome representation is introduced, providing more flexibility to map certain structures that are solutions to a wide range of search problems.

Instead of having a chromosome defined as a sequence of bits, we now define it as a sequence of genes, like the natural counterpart. A gene in turn is defined as a sequence of chromosome positions or loci (usually integer numbers) that code for a particular feature in the phenotype, similar to the natural case. Equation 3.1 below describes the model:

$$G = \cup(g_i^{M_i}) \tag{3.1}$$

In this equation, g_i represents a particular gene, $i = 1, \ldots, n$; and M_i may assume the values 0 or 1. The gene g_i can be thought as an integer or real vector. There is a total of n genes in the chromosome. M is a binary vector that determines the activation state of each gene g_i: if M_i equals 1, then its associated gene, g_i, is activated; and, conversely, if M_i equals 0, then its associated gene, g_i, is inactive. Here, we employ the terms active and inactive genes in a similar sense to the one observed in nature: active genes contribute to the phenotype construction, while inactive genes do not. Thus, we can conceive this chromosome model as a double string, the main one made up of a collection of genes, and an auxiliary binary string, M. We call the former *main string*, whereas the latter is denominated *activation mask*. This is illustrated in Figure 3.1

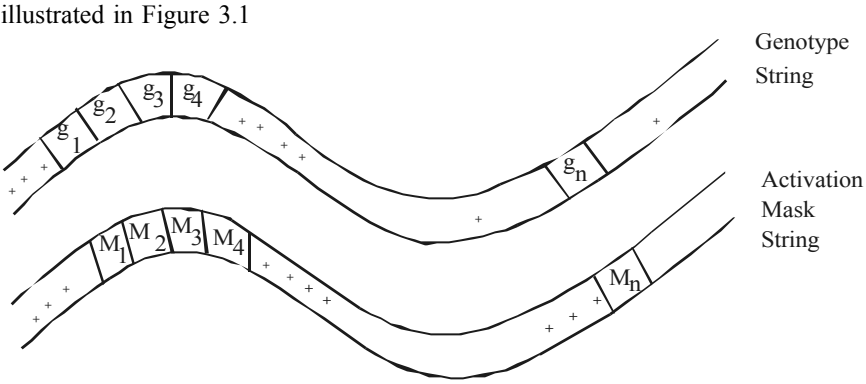


Figure 3.1 Illustration of a chromosome model that implements a Variable Length Representation system.

The chromosome effective length can be defined by the variable m in the following way:

$$m = \sum_{i=1}^{n} M_i \tag{3.2}$$

According to equation 3.2, the chromosome effective length is given by the total number of active genes. We can define another variable, $l = n - m$, which accounts for the total number of inactive genes. The value of n is kept fixed, whereas the values of m and l change along the generations and also within a population.

Through this simple model, the GA is now endowed with the capacity of representing solutions of arbitrary sizes. Instead of fixing the size of the solution of a particular problem, possibly through the use of previous knowledge, the GA has now means to find the size of the desired solution. It is interesting to notice that the total chromosome length is still fixed to n, though the effective length, corresponding to the number of active genes, is now variable. However, it is up to the user to set the parameter n to a value large enough to allow the GA to find the solution without surpassing it. Overall, this is not a problem, since the user often has at least a slight idea of the size of the desired solution.

The basic flow of the genetic algorithm remains unchanged. The fitness evaluation and selection steps are carried out in the same way. Nevertheless, the genetic operators have to be adapted in order to handle two strings per chromosome, the main and the activation ones.

Crossover operators usually splice the contents of two strings to produce new descendents. However, when using the variable length representation, the activation mask also needs to be affected by the crossover operator. If the main string is cut into a particular set of N points, then the activation mask will correspondingly be cut into N points.

The mutation operator performs as in its original version, acting only in the main string of the chromosome.

Besides the crossover and the mutation genetic operators, it is also necessary to introduce a new genetic operator, which is going to manipulate the activation string. This operator provides the means to control the way the GA will sweep different dimensionalities* of the search space. We call this new operator a *Sweep* operator. This operator is applied exclusively to the activation mask in a way similar to the mutation operator: each bit of the activation mask is changed with a low probability, activating or deactivating the corresponding genes. We can devise some sweeping strategies by changing the way this operator is applied:[7] Increasing Length Genotypes Strategy (ILG), Oscillating Length Genotypes Strategy (OLG), and Uniformly Distributed Initial Population (UDIP).[7]

---

* We define the dimensionality D of the search space as its subset that includes all possible solutions represented by chromosomes with an effective length of D.

Increasing Length Genotypes Strategy (ILG)

This strategy has been devised within the context of the so-called *Species Adaptation Genetic Algorithm* (SAGA) methodology, and it seems to be the most suitable one for problems in the domain of engineering. The Increasing Length Genotypes strategy is inspired by the way complex biological organisms are formed from simple ones.[1] In the ILG strategy, all the genotypes are initialized with the same activation mask, in which just a few bits are set to "1;" hence, just a few genes are initially active; simple phenotypes are therefore produced in the beginning of the evolutionary process. Except for the fact that we fix a minimal number of bits set to "1" in the initial mask, its general pattern is randomly initialized. The *sweep* operator may only activate genes in this strategy. As it will be shown, the population tends to grow homogeneously in size, avoiding therefore problems of mating between radically different genotypes, as described by Goldberg[8] (1989).

There are two basic advantages in using the ILG strategy: the implicit control mechanism for the solution sizes and the gain in speed. The implicit control mechanism is characterized by the experimental observation that the average genotype size of the population usually stabilizes at a particular value, instead of growing until the upper bound limit of the genotype. This is due to the fact that, once solutions are found with a particular size, selection will start to check individuals larger than this optimal or sub-optimal size. The gain in speed exhibited by the ILG strategy stems from the fact that solutions with complexity much beyond the one of the optimal solution, whose evaluation is often more time consuming, are not sampled.

Oscillating Length Genotypes Strategy (OLG)

The OLG strategy is a variation of the ILG, in that the genotypes are also allowed to decrease in size. The *sweep* operator now switches, activating and deactivating genes. One possibility is to assign to this operator a rate obeying a sinusoidal function. The main purpose of this strategy is to create pathways from large to smaller genotypes with similar fitness values.

Uniformly Distributed Initial Population (UDIP)

Instead of starting with a population of small genotypes, the initial genotype sizes are now randomly assigned to values ranging from one to the maximum number of genes n. Therefore, the initial population is heterogeneous in terms of genotype size. The *sweep* operator rate is assigned to a constant and low value. The objective of this strategy is to promote the EA to find out the best genotype size from heterogeneous populations.

Of course, it is possible to conceive many different techniques, apart from the above ones. We provide an example of the application of this variable length representation to the problem of the synthesis of passive filters. This class of electronic circuits is used in many areas, such as audio, video, medicine, etc. Further

details are provided in the application chapter of this book, but we advance here some basic features:

- Use of a GA to the synthesis of passive filters from scratch. The chromosome must encode the components' arrangement in the circuit, and the nature and value of each component.
- A particular filter specification, such as low-pass, band pass, or notch,[9] is chosen as target.
- The GA must evolve not only the structure of the circuit, in the way described in item 1, but also its size.

In order to evaluate the sweeping strategies, we need to introduce some performance statistics that consider, as performance criteria, the chromosome fitness and its size. Throughout this section, we show our results using a performance measure that rates fitness and parsimony, as given by:

$$r_{best,j} = \frac{U_T(F_{best,j}) \cdot F_{best,j}}{G_{best,j}} \tag{3.3}$$

$$U_T(f) = 1, \quad if \quad f > T; \qquad U_T(f) = 0, \quad if \quad f \le T \tag{3.4}$$

where $F_{best,j}$ and $G_{best,j}$ stand, respectively, for the fitness and genotype size of the best individual, for a particular execution j of the evolutionary algorithm. $U_T(f)$ is a unit step function over a fitness threshold T selected by the user. If the best chromosome fitness is higher than the specified threshold T, the ratio $r_{best,j}$ defined above will be greater than 0, so that the higher its value, the better the VLR methodology. When taking into account a set of executions, two other measurements can be derived: $(r_{best})_{max}$ and $\langle r_{best} \rangle$ that correspond to the maximum and average values of $r_{best,j}$ over a number of executions of the EA.

The main idea behind these indexes is to give the same emphasis to the circuit's fitness and size. There can be different versions for the r-indexes. For instance, if the circuit correctness is more important than its parsimony, the fitness value in equation (3.3) could be squared.

Table 3.1 compares the performance of the three sweeping strategies when applied to the three cases of filter synthesis, low-pass, band-pass, and notch, resulting in nine experiments. The low-pass filter presents a cutoff frequency at 1,000Hz; the band-pass filter has a passing band between 2,000Hz and 3,000Hz; the notch filter passes the signals at frequencies below 2,000Hz and above 4,000Hz. Each experiment involved 20 GA's executions.

Table 3.1 Comparison of the three sweeping strategies performance in the problem of low-pass (LP), band-pass (BP), and notch filters synthesis. The first column displays the methodologies; the second and third columns present the performance according to the r-indexes previously defined. The fourth column, F_{best}, and the fifth column, G_{best}, respectively represent the fitness and genotype size of the individual that achieved the highest ratio (fitness/size) along all the EA's executions. The three remaining columns provide the mutation rate, success percentage, and the expression of the Sweep operator respectively. In bold we emphasize the best values.

Methodology	**$(r_{best})_{max}$**	**$<r_{best}>$**	**F_{best}**	**G_{best}**	**$\%_{suc}$**	**P_M**	**Sweep**
ILG(LP)	228.6	67.4	914.4	4	70	0.8	$0.2e^{-0.005g}$
ILG(BP)	110.3	18.9	882.3	8	30	2	$0.2e^{-0.005g}$
ILG(Notch)	73.1	26.3	803.9	11	45	2	$0.2e^{-0.001g}$
UDIP(LP)	94.6	70.1	946.2	10	95	0.8	0.03
UDIP(BP)	77.8	35.7	933.9	12	50	2	0.03
UDIP(Notch)	67.6	13.2	878.5	13	20	2	0.03
OLG(LP)	121.5	74.4	972	8	80	2	$0.2\text{sen}(g/20)\, e^{-0.005g}$
OLG(BP)	164.1	23	820.6	5	20	2	$0.2\text{sen}(g/20)\, e^{-0.005g}$
OLG(Notch)	108.2	8.3	865.7	8	10	2	$0.2\text{sen}(g/20)\, e^{-0.005g}$

In terms of the r-indexes, one can see from Table 3.1 that the ILG and OLG strategies achieve the highest $(r_{best})_{max}$ values, whereas the UDIP strategy is slightly better in regards to the average statistics, $<r_{best}>$. Furthermore, the percentage of successful experiments for the OLG strategy, in the case of band-pass and notch filters, is much lower than in the case of the ILG and UDIP strategies. This table also presents the mutation rate and the general expression of the sweep operator in each experiment. We observed that the GA performance is very sensitive to these two parameters. The authors have performed comparative tests to find good values for these parameters.

Figures 3.2, 3.3, and 3.4 show the behavior of the three strategies when applied to the synthesis of the three different filters' specifications, low-pass, band-pass, and notch respectively. These graphs display the average chromosome effective length of the population, averaged over 20 executions, along 400 generations.

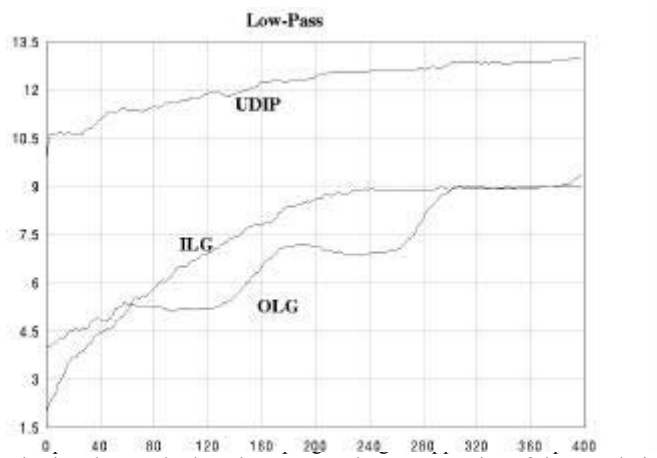


Figure 3.2 low-pass filter synthesis. The graph plots the average chromosome size of the population as a function of the number of generations.

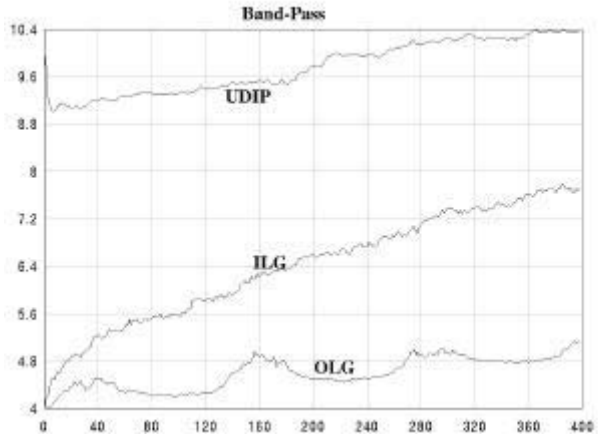


Figure 3.3 Comparison of the three sweeping strategies applied to the problem of band-pass filter synthesis. The graph plots the average chromosome size of the population as a function of the nmber of generations.

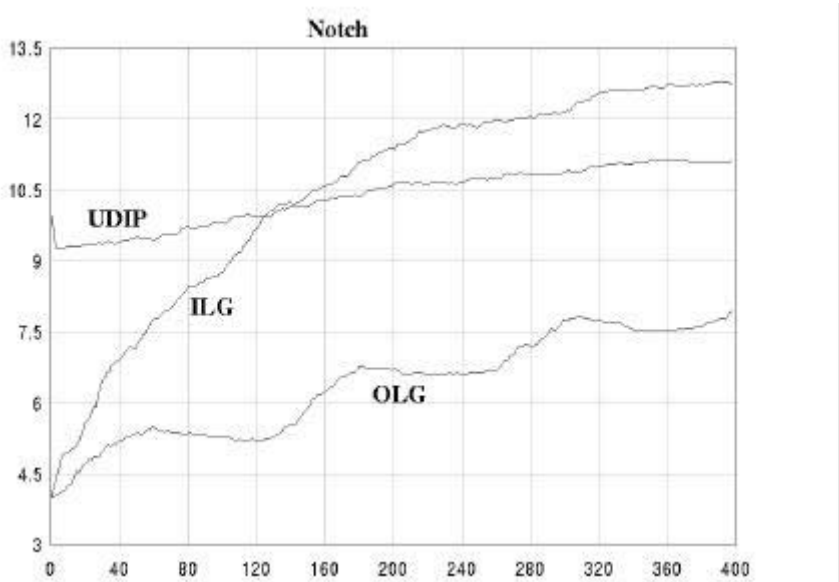


Figure 3.4 Comparison of the three sweeping strategies applied to the problem of notch filter synthesis. The graph plots the average chromosome size of the population as a function of the number of generations.

It can be seen from the graphs shown above that the initial chromosome length for the ILG and OLG strategies is below five genes, meaning that, initially, circuits with less than five components are being sampled. The average chromosome size gradually increases along the generations, in a steady way for the ILG strategy, and oscillating in the case of the OLG strategy. For these two strategies, it can be

observed that the rate of increase in the chromosome length, given by the gradient of the curve, diminishes with the generations. This fact can be clearly observed in the case of the low-pass filter, for which the curves for the ILG and OLG strategies stabilize after the generation 300 in the value of nine genes, meaning that satisfactory circuits with nine components were found.

This allows us to formulate a general conclusion over these two strategies, i.e., whenever better solutions can not be found when the number of components increases, the population will converge, in terms of chromosome size, to smaller solutions. This can also be observed for the OLG strategy in the case of the band-pass filter. In most of the executions, the GA failed to find solutions with chromosome sizes above four.

We did not show details of the fitness evaluation function utilized in these experiments. For now, let's just say that the fitness can take values between 0 (worst) and 1000 (best). This function will have its expression presented later in this book. The graphs of Figure 3.5, Figure 3.6, and Figure 3.7 show the best individual fitness along the evolutionary process, averaged over the 20 executions for the nine experiments, for the low-pass filter, the band-pass filter, and the notch filter, respectively. From these graphs we can see that the UDIP strategy usually achieves best fitness values, and the OLG strategy performance deteriorates for more complex filters' specifications, such as band-pass and notch filters.

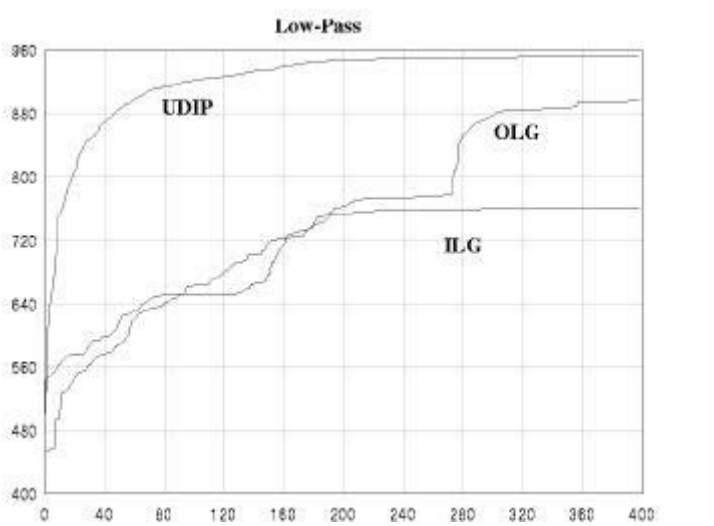


Figure 3.5 Comparison of the three sweeping strategies in terms of fitness for the low-pass filter. The X-axis refers to the generations and the Y-axis represents the fitness.

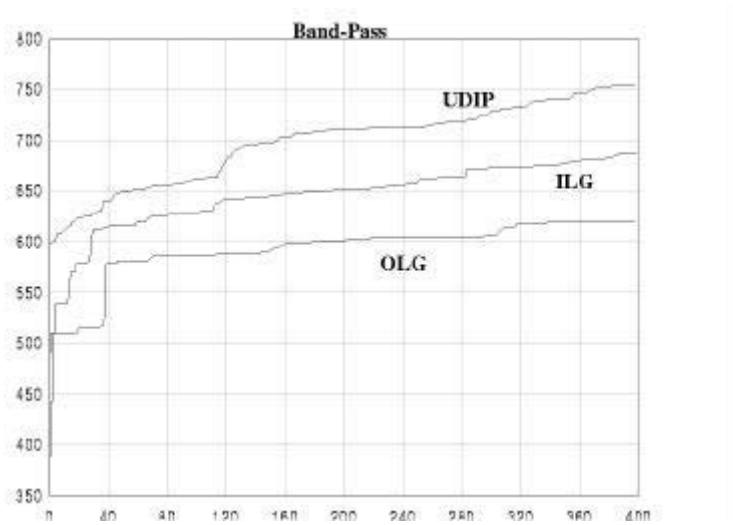


Figure 3.6 Comparison of the three sweeping strategies in terms of fitness for the band-pass filter. The X-axis refers to the generations and the Y-axis represents the fitness.

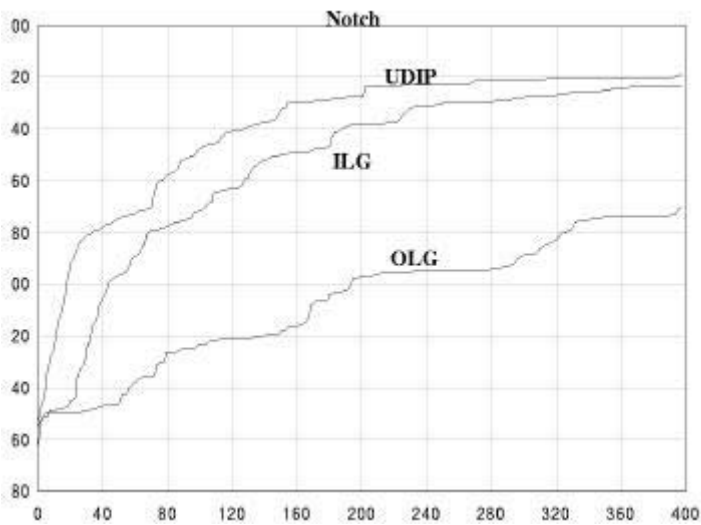


Figure 3.7 Comparison of the three sweeping strategies in terms of fitness for the notch filter. The X-axis refers to the generations and the Y-axis represents the fitness.

3.2 EVOLUTIONARY ALGORITHMS USING THE MEMORY PARADIGM

Within the genetic programming framework, non-coding segments have been studied from the bloating phenomenon perspective, i.e., tree size explosion accompanied by a reduced performance of the optimization process. According to Banzhaf et al.[10] (1997), the advantage of variable size genomes is their ability to choose their own representation, whereas the apparent disadvantage is the production of non-coding segments. They concluded that while introns (non-coding segments) do not affect the fitness of the individual directly, they may affect the likelihood that the individuals will survive, by protecting them against destructive crossover. In another work, Wu and Lindsay[11] (1995) performed an empirical investigation of the influence of non-coding segments in the performance of Genetic Algorithms.

Another implicit role of non-coding segments in artificial evolutionary systems relates to the formation of neutral networks. Harvey and Thompson (1997)[12] investigated this implicit functionality of non-coding segments in the context of . They have used the concept of neutral networks of fitness landscapes in the evolution of FPGA's circuits. A neutral net is a connected pathway through the genotype space which can be transversed through the application of genetic operators such as mutation, while not changing the fitness. These networks are formed if the genetic encoding has a number of potentially useful redundant loci (called junk DNA), which are functionless within a particular context, but given different values elsewhere, they may well become significant. It has been suggested that neutral networks played an important role in the evolution of a two-tones recognizer circuit.[13]

Here, we propose an explicit use of non-coding segments; instead of initializing the genes' contents randomly as they are being activated, we pick out a gene, known a priori to be probably fitter. In order to use this approach, called memory paradigm, our representation will have to be such that the evaluation of each gene as an individual unit will have to be enabled,[14] as explained below.

We start by explaining the idea of memory based evolutionary systems. They process an additional data structure apart from the chromosomes. This data structure plays the role of a gene storage memory, increasing the potential of EAs to solve more complex problems. Chapter 6 shows an application of the memory paradigm in the synthesis of a digital function.

There are many ways to include additional data structures to GAs. The authors suggested and tested a "central genetic memory" that stores genes along the evolutionary process. A genetic memory M with k positions can be defined by:

$$M = \{g_1, g_2, ..., g_k\}$$

where g_1 to g_k are stored genes. It remains to be defined the criteria to access this genetic memory. At the beginning the memory is empty. A particular gene g_i will be stored in the memory if its fitness F surpasses a limit f_c:

$$F(g_i) > f_c$$

where $F(g_i)$ gives the fitness of the gene g_i. The limit f_c depends upon the particular application. As said before, we need to assess not only the performance of the whole chromosome (collection of genes) as in the case of conventional EAs, but also the performance of each gene.[14] For instance, in the case of evolution of digital circuits, as it will be seen in Chapter 6, the representation of chromosomes as sums of products allows the use of the memory paradigm.

When the memory is at least partially filled, it will be accessed throughout the evolutionary process. As explained before, the genetic memory is used in conjunction with the non-coding (inactive) genes observed in VLR systems. When a particular gene is activated (Sweep operator, section 3.1), the new contents of the gene will be retrieved from the memory, instead of being randomly initialized. A position of the genetic memory is selected randomly whenever a non-coding position is activated.

Evolutionary algorithms using the memory paradigm will have the following advantages over conventional ones:

- Overcoming schemata processing limitations[8] of classical GAs, since the highly fit short sized schemata, initially spread throughout the population, are now explicitly gathered to constitute longer sequences of genes

- Saving useful genetic material from being lost once they appear in low fitness genotypes, since they are now stored in the memory. This is akin to the diploid effect of preserving genetic material in natural evolution[8]

The main consequence of the use of the memory paradigm and its associated representation is the departure from the implicit schemata processing concept of traditional genetic algorithms, in the case of building blocks larger than the gene size; though, the implicit schemata theorem still holds at the gene level.

3.3 MULTIPLE-OBJECTIVE OPTIMIZATION

Search problems encountered in the real world are often characterized by the fact that many objectives must be satisfied. Although this topic is called *multi-objective optimization*, we are in fact dealing with a task of achieving acceptable values for a large number of objectives. This concept is described in Fonseca and Flemming,[15] 1995:

> *The simultaneous optimization of multiple, possibly competing, objective functions deviates from the single-function optimization in that it seldom admits a perfect (or Utopian) solution. Instead, multi-objective optimization problems tend to be characterized by a family of alternatives that must be*

> *considered equivalent in the absence of information concerning the relevance of each objective relative to the others. Multiple solutions, or multimodality, arise even in the simplest nontrivial case of two competing objectives ....*

It is increasingly important to use an efficient method to cope with the problem of multiple-objectives optimization, since most of the problems in the area of engineering design, the focus of this book, encompass the satisfaction of many objectives.

Conventional optimization techniques, such as gradient-based and simplex-based methods, were not designed to cope with multiple-objectives search problems, which have to be transformed into single objective problems prior to optimization. On the other hand, evolutionary algorithms are considered to be better tailored to multiple-objectives optimization problems. This is mainly due to the fact that multiple individuals are sampled in parallel, and the search for multiple solutions can be more effective. Of course, this is a rather qualitative argument, which we intend to substantiate with examples later in this section. Notwithstanding, it is the authors' belief that, as described by Fonseca and Flemming[15] (1995), multi-objective optimization is the area where evolutionary computation really distinguishes from its competitors.

This section starts by reviewing some basic approaches utilized in conjunction with evolutionary computation for multiple-objective optimization. Later, we propose a novel technique to handle this problem.

3.3.1 Evolutionary Computation Applied to Multi-Objective Optimization

Evolutionary algorithms typically work with a scalar number to reward individuals' performance, the fitness value. In the case of a single-objective optimization problem, we call this scalar $f(x)$, where x is a particular individual. Considering a multiple-objective problem, we can now define the fitness vector $\boldsymbol{f(x)}$:

$$\mathbf{f(x)} = (f_1(x), f_2(x), ..., f_n(x))$$

where $f_i(x)$ represent the scalar components of $\boldsymbol{f(x)}$. The search problem is now re-stated to the one of seeking for optimal values for all the functions $f_i(x)$. Nevertheless, as mentioned before, this is nearly utopic in the case of real problems, and the actual task is to find acceptable values for all the objectives. We can then refine our notions of local and global optima to the case of multiple-objective problems: local optima are solutions in which only a sub-set of objectives are satisfied (according to a particular specification), while global optima are solutions in which all the objectives are satisfied.

It is not straightforward to incorporate a large number of specifications, which would result in a vectorial measure of fitness into the basic flow of evolutionary algorithms. We will describe three evolutionary computing techniques that have been proposed to tackle the above issue:

- Plain Aggregating approaches
- Population-Based Non-Pareto approaches
- Pareto-Based approaches

3.3.1.1 Plain Aggregating Approaches

This is the most straightforward approach, to transform the objective vector in a scalar. It is simply accomplished by the traditional weighted sum, i.e.,

$$f(x) = \sum_{i=1}^{n} w_i \, f_i \, (x) \qquad (3.5)$$

According to the equation 3.5, the fitness value of the individual x will be given by the sum, over n objectives, of the fitness corresponding to each objective, $f_i(x)$, multiplied by the weight w_i.

The advantageous simplicity of this strategy is nevertheless accompanied by serious drawbacks:

- The correct setting of the weights w is a problem to be solved, requiring usually many runs of the GAs until reasonable values are achieved.
- The absence of interaction with the decision maker can play a harmful role in the search process: it is very common for the GA to arrive at a solution where some objectives are over-satisfied, whereas other ones are under-satisfied.

There are many versions of the plain aggregating approaches: Wilson and Macleod[16] tried to minimize the weighted difference between objective values and corresponding goals, thereby introducing an input from the decision maker to the system; the accumulation of penalty factors to handle constraints has also been proposed.[8] However, this late approach may suffer from the problems that commonly arise when using penalty functions, as discussed in Chapter 1.

In order to overcome the drawbacks intrinsic to the plain aggregating approach, a new technique is proposed later in this chapter.

3.3.1.2 Population-Based Non-Pareto Approaches

This approach has been proposed by Schaffer,[17] calling it Vector Evaluated Genetic Algorithm (VEGA). The main difference of this evolutionary algorithm, compared to conventional ones, occurs during the selection process: in this case, sub-populations are selected from the old generation. Each population is selected according to the

individual's performance regarding one particular objective. The sub-populations are again merged together through the application of the crossover and mutation operators.

The technique can also be viewed from the framework of speciation, which is defined later in this chapter. From this point of view, this method induces the niche formation, where the sub-populations are the different niches in which the whole population is divided. Each niche is a set of specialized individuals; in this particular case, each niche will consist of a group of individuals, good at a particular objective.

This procedure corresponds to an implicit linear combination of the objectives, with its overall effect being somewhat akin to the plain aggregating techniques. In this particular case, the weights correspond to the size of the sub-populations, i.e., the bigger the size of a sub-population, the larger is the value of the implicit weight associated to the objective. Nevertheless, there is a very important feature that differentiates VEGA from the plain aggregating approaches: the value of each implicit weight is also proportional to the inverse of the average fitness, over the whole population, in terms of the correspondent objective. This is accomplished by reducing or increasing the size of a particular sub-population according to the average fitness on the corresponding objective. As explained in Fonseca,[15] 1995:

> *VEGA selection adaptively attempts to balance improvement in the several objectives dimensions, because more good performers in one objective can cause the corresponding average performance to increase and the objective's weight to decrease accordingly.*

This technique is the basis of the strategy presented in the next section.

3.3.1.3 Pareto-Based Approaches

These approaches are based on the concept of Pareto optimality. Given a population of GA individuals, one particular individual v dominates another individual u if and only if:

$$\forall i \in \{1,...,n\}, v_i \leq u_i \qquad \cap \qquad \exists i \in \{1,...,n\}, v_i < u_i$$

where v_i and u_i represent the fitness value referring to objective i for the individuals u and v respectively. In this particular example, there are n objectives that have to be minimized. Therefore, one individual dominates another if it is better in at least one objective, and it is not worse in any of the remaining objectives. The so-called Pareto-Optimal set is constituted by all the individuals of a particular generation that are not dominated. Based on this definition, many evolutionary methods have been proposed.

Goldberg[8] proposed a mechanism in which all the nondominated individuals were removed from the population and given the same probability of reproduction. A new set of dominated individuals was then found over the remaining individuals,

and a lower chance of reproduction was assigned. This process went on until no nondominated set could be extracted. Fonseca and Flemming[18] proposed a scheme in which the individual's rank (using rank selection) in the current population corresponds to the number of individuals by which it is dominated. Many authors have also proposed the use of tournment selection based on Pareto dominance.[19,20] In this case, one of the Pareto based rank schemes described earlier is used as a basis to decide the winner of a binary tournment.

According to some authors, the main advantage of the Pareto based approaches stems from the fact that nondominated individuals are equally treated, and, as a consequence, solutions that perform better in many of the objectives' dimensions are more likely to be found. However, the main caution when using Pareto based techniques is that they can fail whenever trying to find a compromise among many objectives, as a pure Pareto technique is blind regarding user preference information. This is explained in Fonseca,[15] 1995:

> *pure Pareto EAs cannot be expected to perform well on problems involving many competing objectives. And may simply fail to produce satisfactory solutions due to the large dimensionality and size of the trade-off surface. As the number of actually competing objectives increases, more and more of the search space can be expected to conform to the definition of Pareto optimality, which makes the "theoretical" problem of finding nondominated solutions easier! Unfortunately, in the total absence of preference information, the EA will face the 'impossible' task of finding a satisfactory compromise in the dark, which can only occur by pure chance.*

3.3.2 A New Approach for Multi-Objective Optimization: Energy Minimization Strategy

We present here a strategy formulated by the authors to tackle multi-objective optimization. This new approach aims to overcome the limitation of other methods. This strategy derives from the plain aggregating approach, in the sense that the individual fitness *f(x)* is defined as a weighted sum:

$$f(x) = \sum_{i=1}^{n} w_i \, fn_i(x) \tag{3.6}$$

In the equation 3.6, the term fn_i accounts for the normalized value of the fitness associated to the objective i, which attempts to compensate the fact that different objectives may have different natures and thus magnitudes. As a way to equalize the contribution of each objective in the fitness expression, we defined the normalized fitness value in the following way:

$$fn_i = \frac{f_i}{\bar{f}_i} \tag{3.7}$$

where the denominator represents the average of the fitness values scored by the individuals with respect to objective i. This formulation tries to avoid disproportionate contributions of the objectives in the aggregating equation (3.6).

Though simple, this normalization is an essential step to this method. The problem of setting the weights remains to be solved. The simplest approach would be to choose equal weights and have all the objectives contributing evenly to the fitness equation. Nonetheless, this would often make the evolutionary algorithm converge to a local optima. The notion of local optima for the multi-objective case corresponds to a situation where most objectives achieve satisfactory values (being usually over-satisfied), whereas the most difficult ones are not fulfilled. Of course, the conventional way used to sort out this problem was to set values for the weights according to the current knowledge on which objectives are more difficult to be satisfied.

However, this kind of previous knowledge is usually very imprecise. It is desirable to have the GA finding adequate values for the weights, and, most importantly, adaptively changing them according to the current level of satisfaction of the objectives.

Generally speaking, we need weights that are increased proportionally to the inverse of the satisfaction level of an objective, so that poor performance objectives dominate the aggregating equation. Based on this requirement, we devised a weight updating scheme similar to the one employed in one of the Artificial Neural Network models, the Backpropagation. The weight updating obeys the following equation:

$$w_{i,t+1} = k_1 \alpha \cdot w_{i,t} + k_2 \cdot (1-\alpha) \cdot e_{i,t} \tag{3.8}$$

where $w_{i,t+1}$ is the weight associated to the objective i in the cycle t of the evolutionary algorithm; $e_{i,t}$ is the system error to satisfy a particular criteria i; and k_1, k_2 and a parameters which will be explained later.

For the readers that are not acquainted with Artificial Neural Networks (ANN), we shall give a brief explanation of what is being expressed by the equation (3.8). Artificial Neural Networks encompass a class of algorithms that use some concepts of the animal nervous system as metaphors, in the same way Evolutionary Algorithms are inspired in natural evolution. Particularly, the issue of learning is of awesome importance to this area: we can usually define learning in this class of artificial systems as the process of adjusting a set of weights until a particular behavior is acquired. The target behaviors are usually defined in pattern recognition or time series forecasting applications. The weights are real numbers that connect basic processing units, also called artificial neurons. The above equation (3.8) is based on one of the most successful ANN learning algorithms, the Backpropagation (BP). This algorithm tries to adjust the weights w in such a way to reduce the error e between the actual and the desired ANN.[21]

Coming back to our problem of multiple-objectives optimization, it is clear that we

are also trying to minimize an error, the one between the users' specification for the *n* objectives and the values achieved by the GA. Therefore, roughly imitating the BP algorithm, we can attempt to adjust the weights in such a way to reduce the error achieved by the system regarding all the objectives. This simple procedure is then explained.

The weights and the error variables shown in equation (3.8) take two indexes, *i* and *t*: the former points to a particular objective, and the latter points to a particular generation of the evolutionary process. *k1* and *k2* are normalization constants, whose values are determined by a procedure that is described later. Focusing only on the second term of equation (3.8), we can see that the higher the error to satisfy a particular criteria *i*, the higher the associated weight will be in cycle $t+1$. There are many ways to define the error to satisfy a criteria or objective, and we propose the satisfaction of the overall system error:

$$e_{i,t} = \left| \frac{User_i - \bar{f}_{i,t}}{User_i} \right| \tag{3.9}$$

This statistic considers the percentage of the difference between the user specification for a particular objective i , $User_i$, and the population average fitness for objective *i*, at the generation (or instant) *t*. This procedure increases the influence of the objectives that are further to its satisfaction in the weight updating equation.

One can then ask what is the purpose of the first term in the weight updating equation: the idea is very much the same as a similar term used in the BP algorithm, known as *momentum*, which gives more stability to the process. This term implements a simple memory of the last value taken by the weights, and, by doing this, drastic changes in the weights' values are avoided. These drastic changes could hurt the evolutionary process that could quickly switch between solutions where a subset of objectives are over-satisfied and the others are under-satisfied. As these changes are not smooth enough, the GA will probably miss global optima. Finally, the constant α employed in equation (3.8) balances the contribution of the two parcels, the memory one and the error proportional one. The higher the value of α, the slower the change in the weights will be. According to the normalization procedure we adopt, this parameter can take real values between 0 and 1.

Having explained the basic idea of this procedure, we are now going to examine its application in more detail. The starting point is to choose the weights' initial values, w_{i0}. These values are arbitrary, but they may be made equal to one another, unless some additional information about the relative difficulty to fulfill an objective is available. We can then define the initial cumulative value of the weights, S_{w0}:

$$S_{w0} = \sum_{i=1}^{n} w_{i,0} \tag{3.10}$$

At generation 0, the fitness of each individual is evaluated according to equation (3.6), using the initial weights' values. After a particular interval of generations, the weights are changed according to equation (3.8). This interval is also empirically determined, and it should be enough for the GA to "react" to each new set of

weights.

Whenever the current generation lies in the update interval, equation (3.8) must then be used. Beforehand, the errors $e_{i,t}$ and their cumulative sum, $S_{e,t}$, are calculated (equation (3.9)). We then establish the weights' sum $S_{w,t}$ before the individual weights are actually calculated. This is accomplished by the following equation:

$$S_{w,t} = k_3 \sum_{i=1}^{n} e_{i,t} = k_3 S_{e,t} \tag{3.11}$$

where:

$$k_3 = \frac{S_{w0}}{S_{e0}} \tag{3.12}$$

where $S_{e,0}$ is the cumulative error at generation 0. What is then the objective of this apparently rather arbitrary procedure? We just wanted to make the cumulative value of the weights proportional to the cumulative value of the errors, and the reason for this will soon become clear. The constant k_3 is simply a way to account for the initial value of S_{w0} chosen by the user. It should become clear that k_3 is calculated only once, in the beginning of the GA, whereas $S_{w,t}$ is calculated whenever equation (3.8) is going to be used. But, in order to finally update the weights, we must determine the values of the constants k_1 and k_2. They are calculated in such a way to guarantee that the cumulative value of the weights will be the one shown in equation (3.11). After a rather straightforward calculation, we arrive at the following values:

$$k_1 = \frac{S_{w,t}}{S_{w,t-1}} \therefore k_2 = \frac{S_{w,t}}{S_{e,t}} \tag{3.13}$$

for t greater than 0.

Overall the mechanism assures that:

- The cumulative value of the weights will be proportional to the cumulative value of the errors (equation (3.11))
- Higher weight values are assigned to objectives that are further from their satisfaction
- The process can be made as smooth as it is necessary

The first statement is clear since we would like the optimization process to be mainly driven by unsatisfied objectives. This statement is accomplished by equation (3.11). Nonetheless, this should be made without losing control of the fulfilled objectives. This is expressed by the third statement: if the process can be made sufficiently gradual, it will be easier to accommodate all the objectives. This is accomplished by the first part of equation (3.8). In order to evaluate how well the

methodology performs, we employ a statistics of convergence of the overall system. The cumulative value of the weights will provide this statistic. Such a statistic can be defined by:

$$E = \sum_{i=1}^{n} w_i^2 \tag{3.14}$$

This is a standard way of defining an energy measure in Boltzmann systems. These systems accomplish optimization by reducing a scalar quantity, usually referred to as energy. What is the meaning of this quantity within this methodology? We can replace the value of w_i by the one shown in equation (3.8). If we expand the final expression, we will have:

$$E = \sum_{i=1}^{n} (k_1^2 \alpha^2 w^2{}_{i,t-1} + k_2^2 (1-\alpha)^2 e_{i,t}^2 + 2 \cdot k_1 k_2 \alpha (1-\alpha) w_{i,t-1} e_{i,t}) \tag{3.15}$$

We can say that the evolutionary system is performing well if the energy of the system is decreasing. According to equation (3.15), an obvious condition for lowering the energy E is the decrease in the error associated to each objective i. However, the energy takes also into account the previous value of the weight associated to each objective, i.e., the lower the weight's value, the lower the energy is. This is a way to take the stability of the optimization process in consideration, since the previous value of the weight reflects the previous error associated to a particular objective. A stable evolutionary system should present the energy function monotonically decreasing along the process. Naturally, the ideal condition is usually hard to attain in the case of real problems.

This methodology has been applied to a variety of problems in . In Chapter 4, for instance, this technique is extensively applied to VLSI optimization of analog circuits. In subsequent chapters, other problems in analog design also benefit from this method.

3.4 SPECIATION IN EVOLUTIONARY ALGORITHMS

In natural systems, speciation is characterized by the reproductive isolation between two groups. A geographic isolation between the groups usually occurs prior to the reproductive isolation. If two groups of the same species live for a very long time in different environments, they will possibly adapt to survive to their particular environments, and subsequently phenotypic differences will emerge.

There have been attempts to model this feature in the context of artificial systems. Nonetheless, the underlying question is: which kind of advantage speciation would confer to artificial systems? In nature, speciation seemed to be a necessity, since each organism must adapt to a particular environment. So, the first question that we need to address is: how can we define environment in the case of artificial systems? The response to this question is that the fitness evaluation function defines the environment. An individual that scores well on a particular fitness evaluation function is well adapted to the "environment" defined by this function.

If we suddenly change the fitness evaluation function, we will force the individuals to adapt to it.

Therefore, our approach to model natural speciation is to carve up the search space in sub-spaces with slightly different fitness evaluation criteria. The main benefit of this strategy is that unmanageable search spaces can be decomposed, facilitating thus the search process.

There are many ways to implement this strategy, and we present now one approach. Supposing that K is the alphabet cardinality and each gene spans r chromosome loci. We can then establish the following indexing:

$$I(g_j) = \sum_{i=0}^{r-1} L_i K^i \tag{3.16}$$

where L_i accounts for the value of locus i of gene g_j. If we are talking about an integer representation, then there will be K^r different genes, each one associated to a particular index $I(g_j)$. The search space can be partitioned in accordance to these indexes. If we wish to create S regions, the following rule may be applied:

$$\frac{K^r}{S}(j-1) \le R_j \le \frac{K^r}{S} j, \qquad j = 1,2,...,S$$

where each region R_j includes the sub-set of genes whose indexes are constrained to the inferior and superior limits defined above. S evolutionary algorithms are executed in parallel to sample the respective regions. It can be verified that this decomposition is performed at the genotype level, whereas natural speciation occurs at the phenotype level, through adaptation.

There are S different fitness functions, F_1, F_2 until F_S, to each particular region. The functions have the following general form:

$$F_i = F_{problem} - P_i, \qquad 1 \le i \le S \tag{3.17}$$

where $F_{problem}$ is the actual fitness function of the problem and P_i is a penalty term. The penalty term is designed to induce the correct distribution of the sub-populations throughout the sub-spaces. When we start running the EAs, we will have S sub-populations randomly initialized; therefore, we expect that each EA will start sampling a sub-population that is uniformly distributed over the search space. But we want that each EA focus on a respective region of the search space according to the defined indexing system. Thus, the penalty terms P_i will strongly reduce the fitness of the chromosomes containing genes whose indexes are outside the region boundaries of the associated EA. After some generations, the sub-populations will be distributed according to the regions the respective EAs are sampling.

In order to exemplify this indexing scheme, suppose a hypothetical problem in which the cardinality alphabet is four, $K = 4$, and the genes are made up $r = 5$ loci. Thus, there are 4^5 or 1024 genes in the search space. For instance, if we carve up the search space in eight regions and the gene $(2,3,1,0,1)$ is sampled by the EA_3 (region R_3), we can verify if the gene was supposed to be in this region:

$$\frac{1024}{8}\cdot(3-1)\le R_3\le\frac{1024}{8}\cdot 3\Rightarrow 256\le R_3\le 384$$

$$I(g) = 2\cdot 4^4 + 3\cdot 4^3 + 1\cdot 4^2 + 0\cdot 4^1 + 1\cdot 4^0 = 721$$

As depicted by the expressions above, this gene does not belong to R_3. Therefore, the penalty term P_3 will hold. According to its index, this gene should be sampled by EA_6, and P_6 would therefore not hold for this gene.

The natural question one can ask at this point could be, is this procedure useful to solve practical problems? This method is particularly useful in the case of multimodal problems. As described earlier, Multimodal problems present many solutions of interest. According to the representation employed to multimodal problems, the solutions may be represented by chromosomes that are distant at the genotype level. In this case, the above method will be rather helpful. Suppose that a problem presents the fitness landscape shown in Figure 3.8.

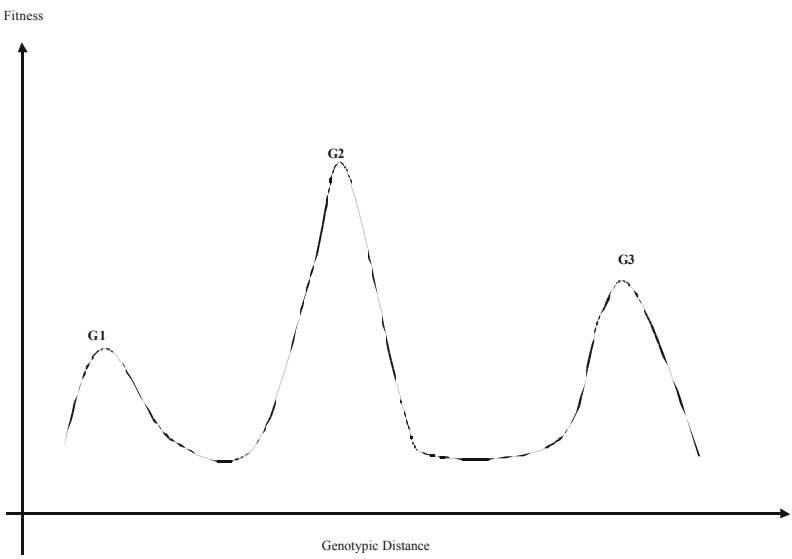


Figure 3.8 Hypothetical fitness landscape for a multimodal problem.

In the hypothetical multimodal problem exemplified by Figure 3.8, the solutions of interest are encoded by the chromosomes G_1, G_2, G_3. Even though G_2 is the fittest, it is not the only solution of interest. If the genotypic distance between these chromosomes is big enough, then a single EA will be more likely to converge to a single solution, either G_1, G_2, or G_3, rather than finding all of them. In this case, our speciation method is very efficient, since it will provide a more uniform sampling of the search space. In the application chapters, we will give a practical example of a multimodal problem in the domain of digital electronics.

3.5 DECEPTION AND EPISTASY

According to the Schema theorem and with the Building Blocks Hypothesis, low-ordered schemata with fitness higher than the population average fitness have more instances from one generation to another. Therefore, the building blocks of the optimal point are likely to stand better chances throughout evolution. These building blocks have generally higher fitness than the average because of the influence of the fitness of the optimal point. However, when the fitness of schema which instances the optimal point is lower than that of other primary schemata, competitors of the optimal schema, it is said that deception occurs. In other words, deception takes place when a successful schema of a primary competitor of order k, with $k<l$ (l is the chromosome length), is not part of the optimal solution. The order of a schema is the number of its specific positions, the ones different of don't care (*).

Because the occurrence of deception is related to a primary competition of order $k<l$, the simplest possible deception problem uses 2 bits. Goldberg[8] has defined the Minimal Deception Problem (MDP) through the following example:

The chromosome 11 represents the optimal solution. Therefore, f(11)>f(10), f(11) > f(01) and f(11) > f(00). If there is no deception, the primary competitions of order 1 must obey the following relationships: $f(1*)^3 \geq f(0*)$ and $f(*1)^3 \geq f(*0)$. If at least one of these relationships is not verified, deception occurs. For 2 bits it is impossible that both relationships do not occur and chromosome 11 continues to represent the best solution. In this way, Goldberg chose the first relationship to be denied, and, based on it, has created two models of Minimal Deception Problems (MDPs).

The denial of the first relationship considers the average fitness of all the solutions belonging to the specific schema:

$$f(0*) > f(1*) \Rightarrow \quad \frac{f(00)+f(01)}{2} > \frac{f(10)+f(11)}{2}$$

Assuming that all the values are positive and 11 represents the optimal point, the above expression can be written as:

$$f(00) + f(01) > f(10) + f(11) \Rightarrow f(01) - f(10) > f(11) - f(00)$$

However, f(11) – f(00) >0, therefore f(01) – f(10) >0, which implies that f(01)>f(10) and f(00)>f(10).

To complete the problem definition, it is necessary to know the relationship between f(00) and f(01). For each relationship, Goldberg has created a MDP. In the first case, f(01) > f(00) and the relationship between the fitness of all individuals is f(11) > f(01) > f(00) > f(10).

For the second case, f(00) > f(01), and the relationship between the fitness of all individuals is f(11) > f(00) > f(01) > f(10).

The graphical representation of the two MDPs can be seen on Figure 3.9.

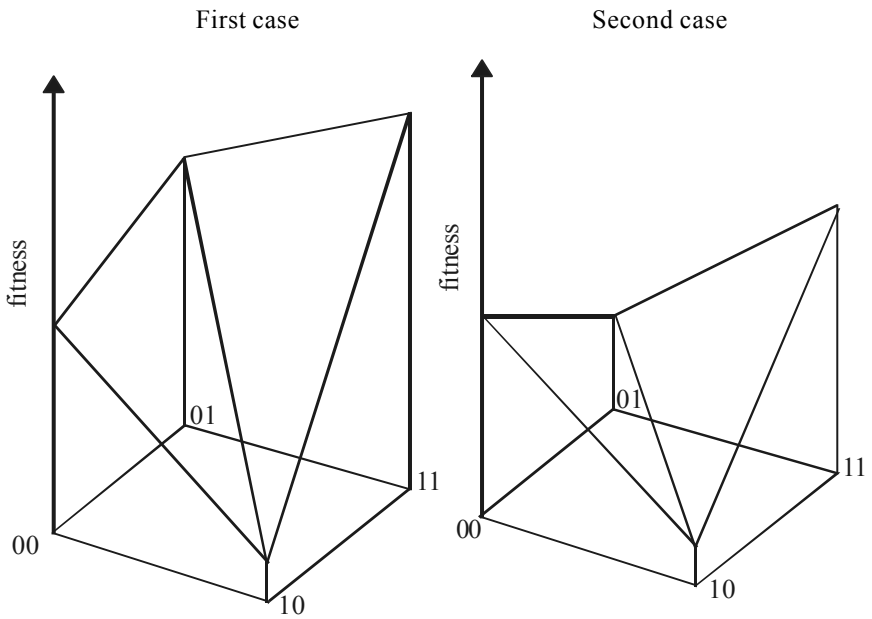


Figure 3.9 Minimal Deception Problems: 1st case f(00) > f(01); 2nd case f(00) > f(01).

Goldberg used GAs to solve both MDPs. For the first case, if the initial population has the same number of all possible chromosomes, the schema 01 starts increasing its number, but when the presence of schemata 00 and 10 decreases, the competition concentrates on the schemata 01 and 11, and the GA converges to the optimal solution.

For the second case, with the same initial population, a similar process occurs and the GA converges to the optimal solution. However, when the initial population has more instances of the sub-optimal point than of the other possible points, it ends up dominating the search process and the GA converges to the sub-optimal point.

Goldberg's experiments have showed that, although the MDPs cause deception, they are not difficult to be solved by GAs ("GA-hard"), and their deception level is limited.

Deceptive problems can be classified as:

- Fully deceptive
- Consistently deceptive
- Partially deceptive

A problem of order l is fully deceptive when all the relevant hyperplanes of order k, with $k < l$, direct the search to the same solution different from the global optimal. This means that all the primary competitions with order lower than l that represent competitors of order l (relevant) are won by schemata of the same point different from the optimal point. This sub-optimal point is known as the deceptive attractor.

A problem of order l is consistently deceptive when no relevant hyperplane of order k, with $k < l$, directs the search to the optimal point, but not all primary competitions of order k are won by schemata that form the deceptive attractor.

A fully deceptive problem is always consistently deceptive, but the opposite is not true. According to Whitley[22] (1992), these classifications can be mathematically defined as:

Fully deceptive $\forall(X,Y)\left\{f(H_{d,x}) > f(Y) \mid Y \in X, Y \neq H_{d,x}\right\}$

Consistently deceptive $\forall(X,Y)\,\exists(Y)\left\{f(H_{g,x}) < f(Y) \mid Y \in X, Y \neq H_{g,x}\right\}$

where X is a set of relevant primary competitors, d refers to the deceptive attractor, and g refers to the global optimal point.

When a problem is deceptive but it is neither fully deceptive nor consistently deceptive, it is considered partially deceptive. Whitley proposes two types of partially deceptive problems: chronically deceptive and fundamentally deceptive.

A problem of order l is chronically deceptive when most of the primary competitions of order k, with k< l, are won by schemata that are not part of the optimal point. This means that at least one winning hyperplain of order k contains the optimal point.

A fundamentally deceptive problem is a partially deceptive problem in which the optimal point cannot be inferred by primary competitions of order l.

Das and Whitley (*apud* Whitley, op.cit.) state that, in case there is no deception, the optimal value of a problem can be discovered based on the primary winners of order l, which form together the desired point. They say therefore that "(...) the only problems that offer challenge to the optimization with GAs are those with some degree of deception."[22] This statement is very polemic because it is based on a static vision of GAs, not taking into account, for example, the population variance that can cause sample errors.

According to Deb,[19] the study of deception can be useful in understanding the dynamics of GAs. The study has been concentrated in three principal points:

- The creation of deceptive functions
- The understanding of the effects of deception in the solution of GAs
- The adaptation of GAs in order to solve deceptive functions

Examples of the creation of deceptive functions and the understanding of its effects are the creation of MPDs and the study of the GA's behavior to solve it. A tool used for this study is Walsh's coefficients. One of its advantages is the association with the idea of schemata through the Walsh-Schema Transform.

3.5.1 Epistasy

The term epistasy has its origin in biology and, as it relates, means the *"(...) functional interaction of non-allelic genes."* The genes are considered alleles when they are responsible for the determination of the same characteristic. When a gene that is not responsible for a certain characteristic influences it, it is called epistatic.

GAs use the biological idea of epistasy to characterize the influence of a gene on another in the representation of the solutions in a problem. This implies the existence of interdependency between the genes of the representation.

The degree of dependency between the genes of a representation is of great importance to the convergence of GAs. Based on the Hypothesis of the Building Blocks, schemata of higher order and with higher fitness than the average are obtained from schemata of lower order. In case the degree of interdependency between the genes is high, the schemata of lower order do not provide significant information. A significant schema has to represent the dependent genes; therefore, the process of building the blocks has to start from schemata of higher order.

Based on these observations, the GA problems can be divided into three cases:

- Problems without epistasy
- Problems with medium epistasy
- Problems with strong epistasy

The problems without epistasy are those in which the genes are linearly independent. These problems usually do not provide difficulties to be solved, and GAs can solve them without difficulties as well. In this case, the use of simpler techniques (greedy) is enough to solve the problems.

The problems with medium epistasy are the most adequate for the use of GAs. The quantification of the medium degree of epistasy is not mathematically precise, but it is related to the ratio between the size of the basis of epistasy and the length of representation. The size of the basis of epistasy takes into account the number of interdependent genes and their arrangement in the representation. The number of interdependent genes is related to the concept of order of a schema, and their arrangement in the representation is related to the concept of length of a schema. When these concepts can not be applied to a significant portion of a chromosome's length, it is a case of medium epistasy. Davidor[23] has created a quantification for the degree of epistasy of a representation for a certain problem. He named it the

epistasy variance.

Problems with strong epistasy are similar to random search, as the bases formed by interdependent genes determine search spaces that are nearly as large and complex as the whole search space of the problem. These problems are not adequate for GAs because they cannot evaluate correctly the schemata of lower order and direct the initial search to more promising regions of the space. Therefore, GAs cannot form solutions of higher order. Because the only schemata correctly evaluated are those of higher order, equal to the degree of interdependency between the genes, the GAs behave as a random search.

Epistatic functions

Some functions have a strong characterization of epistasy. Among others are the NK functions (Weinberg, s.d.), the "Royal Road" functions,[24] and a function known as "scorpion tail."[25]

The NK functions are represented by chromosomes with N positions, in which each gene is interdependent to other k genes. The determination of the blocks of related genes is not rigid and can be done randomly, or it may be conventionalized that the dependency occurs between close genes. For example, if $k = 3$, the dependency of each gene occurs with the two genes at its left and one at its right, or the opposite may happen. Weinberg (s.d.) states that *"(...) it is surprising to notice that the local characteristics of the search space are not sensible to how the positions of the block of k+1 bits are chosen."*

The reckoning of a solution is given by the sum of the values of the dependent blocks of each gene divided by the chromosome length

$$f(x) = \left(\frac{1}{N}\right)\sum b(xi) \tag{3.18}$$

In equation 3.18, $b(x_i)$ is the value of the block of interdependent genes to the gene of position i.

The following is an example of a NK function, with $N = 5$ and $K = 2$, where the dependency relationships occur with the neighbor genes.

Given the following relationships:

000 = 1	001 = 7	010 = 4	011 = 8	100 = 6	101 = 2	110 = 5	111 = 9

the point 10100 is evaluated as

$$(f(101) + f(010) + f(100) + f(001) + f(010)) / 5 = (2 + 4 + 6 + 1 + 4) / 5 = 3.4$$

If one position is changed, the evaluation is changed in $k + 1$ sum factors. For example, the point 11100 would have as evaluation

$$(f(111) + f(110) + f(100) + f(001) + f(011)) / 5 = (9 + 5 + 6 + 1 + 8) / 5 = 5.8$$

On the other hand, the "Royal Road" functions (RR) are represented by chromosomes of N positions divided in blocks of size b. These functions evaluate the individuals according to the number of blocks of value 1. They can be of two types: RR1 or RR2.

The RR1 function takes into account only one level of blocks. The evaluation is value b multiplied by the number of blocks of 1s. The RR2 function considers more than one level, rewarding neighbor blocks of 1s with a value proportional to the set (2^i b). The neighbor blocks are only considered if the position of the first block is a power of 2.

For example, the RR1 function applied to a chromosome of 12 positions and 4 blocks of size 3 would have as optimal point

111 111 111 111

Each block is rewarded with 3, therefore f(optimal) = 3 + 3 + 3 + 3 = 12

For the chromosomes A, B, and C:

A = 101 110 011 110 f(A) = 0 + 0 + 0 + 0 = 0 (no block has 1s exclusively).
B = 000 111 000 001 f(B) = 0 + 3 + 0 + 0 = 3
C = 111 111 111 100 f(C) = 3 + 3 + 3 + 0 = 9

The RR2 function applied to a chromosome of 12 positions and 4 blocks of size 3 would have as optimal point 111 111 111 111.

For the first level, each block is of size 3, therefore f_1(optimal)= 3 + 3 + 3 + 3 =12

For the second level, each block is of size 6. The same chromosome can be seen as

optimal = 111 111 111 111 and f_2(optimal) = 6 + 6 = 12

For the third level, each block is of size 12. The chromosome can be seen as

optimal = 111 111 111 111 and f_3(optimal) = 12

The evaluation is the sum of the partial evaluations for each level:

f(optimal) = f_1(optimal) + f_2(optimal) + f_3(optimal) = 12 + 12 + 12 = 36

Likewise, for the other chromosomes:

A = 101 110 011 110 f(A) = (0 + 0 + 0 + 0) + (0 + 0) + (0) = 0
B = 000 111 000 001 f(B) = (0 + 3 + 0 + 0) + (0 + 0) + (0) = 3
C = 111 111 111 100 f(C) = (3 + 3 + 3 + 0) + (6 + 0) + (0) = 15

The function "scorpion tail" is represented by a chromosome of 20 positions, divided in two blocks. The first block is the x and contains the 15 first positions. The second block is the y, with the other 5 positions. Considering u(x) equal to the number of 1s of block x, and z equal to the number of 0s of block y, the "scorpion tail" function has the general form f(x,y) = u(x) + z(y), unless the whole chromosome consists of 1s. In this case, the function has value 30:

$$f(x,y) = \begin{cases} u(x) + z(y), & if\ \left(u(x) < 15\ and\ z(y) \neq 0\right) \\ 30, & if\ \left(u(x) = 15\ and\ z(y) = 0\right) \end{cases} \tag{3.19}$$

The Analysis of Epistatic Functions

The analysis of epistatic functions involves:

- The verification of the idea of basis of epistasy for the exposed problems
- The presentation of the idea of "stepping stones"
- The problems of hitchhiking
- The presentation of the idea of "(un)correlated landscape" (*apud* Harvey,[6] 1993)
- The relationship between epistasy and deception

The idea of basis of epistasy is linked to the idea of a block and can be clearly seen in the example functions. The basis is the smallest group of genes necessary for the GA to obtain significant information for the discovery of promising regions of the search space of a problem, whatever gene is chosen. The complement "whatever gene is chosen" means that, even if there is an isolated gene that provides this type of information, if other genes are interdependent, these are the ones that determine the basis.

For the NK function, the order of the basis is clearly $K + 1$, as the function itself is based on the sum of the values of the bases. For the RR function, the basis also corresponds to the order and size of the block. However, the "scorpion tail" function has different characteristics. Its analysis can be divided in two parts.

At first, it is verified that the part of the function that evaluates the *x* block can be considered unique, as long as the second condition turns into u(x) + 15. As the function for the *x* block is unique, it is possible to separate the function in two subfunctions: one for the *x* block and the other for the *y* block. As the function for the *x* block is easily optimized, the value of the optimal point is determined in order to join it to the optimal point of the *y* block.

The second part of the analysis refers to the study of the function for the *y* block. This part of the function can be seen as f(y) = z(y), if z(y) > 0, and f(y) = 15, if z(y) = 0. An overview of the whole search space indicates deception for the orders 4, 3, and 2. In case the optimal point (1111) does not appear in the initial population, the fundamental condition of deception (first order) is also verified for the population. In this case, the convergence occurs quickly to the sub-optimal point (0000). If the optimal point is proportionally distributed in the initial population and no elitist techniques are used, the convergence should also occur to the local optimal. Only if the optimal point has predominance in the initial population will the convergence occur to the optimal point.

The behavior of the *y* portion of the function resembles that of the function "without deception, but difficult" of Grefenstette, although the other one causes deception. In this way, this problem has a basis equal to the *y* block, which

corresponds to the whole search space of the difficult part of the function: the *y* portion. That is why this function is considered very difficult for GAs.

The idea of "stepping stones" refers to the *"... schemata of intermediate order and higher fitness due to the combination of schemata of lower order, which can be combined to produce even fitter schemata."*[27] It is interesting the association of this idea with the influence of the basis of epistasy. If the basis is too large, there aren't any smaller "stepping stones" to perform the building process to its order, and the opposite may even take place. The basis should give the dimension of the "stepping stone" according to the associated genes. This is the idea that inspires the RR functions.

The "hitchhiking" problems were verified with the RR2 functions.[27-25] The "Royal Road" functions have received this name because they offer a facilitated way for the construction of schemata of higher order. This facilitation must be offered by the "stepping stones" constituted by the blocks. It would be expected that this facility would grow from the RR1 to the RR2 function, as the "stepping stones" of the latter were apparently reinforced. However, a phenomenon similar to the collateral convergence has taken place. The reinforcement to the value of certain schemata is so much that it makes the strength of other schemata instanced by the optimal point not enough to generate an association between them. It can be said that, with it, the positions that should be substituted by the schema that contains the optimal point are maintained in the reinforced schemata, hitchhiking with it.

Another surprise of the RR functions is the observation that the performance of a specific "hill climbing" method is superior to that of the GA. This made Mitchell[26] question *"when genetic algorithms would surpass hill climbing"* and made Holland[27] create more elaborated versions of the RR function, for which the performance of the GA is superior. These studies are important for a better understanding of what characterizes the most adequate type of problem to use GAs.

The idea of "(un)correlated landscape" explains how the search space of a problem of certain representation is structured. A region of the space is considered a "correlated landscape" when the change in one bit leads the result of a solution to a position close to that of the former/earlier result. In an "uncorrelated landscape" the distance between the solutions different in only one position can be very diverse. In general, small distances in the Hamming space correspond to small differences in the evaluation for "correlated landscapes" and unpredictable differences for "uncorrelated landscapes." The space that represents epistatic problems is considered "uncorrelated landscape." The idea of "(un)correlated landscape" has been used in the attempt to foresee the evolution of a GA.[24]

The relationship between epistasy and deception can be seen, above all, in the meaning of a basis of epistasy and Walsh's coefficient of higher order with the value different from zero. Both cases refer to the interdependency between genes. Every deceptive problem is also epistatic, and the idea of basis can be applied to deceptive problems as well. However, not every epistatic problem is deceptive, as can be verified with the RR functions.

A Quantification of Epistasy

Davidor[23] proposed a quantification of epistasy. It is based on the difference between the observed value of a chromosome and the expected value determined by the sum of the value of the genes. This can be calculated by the following routine:

First, the average evaluation of an allele is given by:

$$A_i(a) = \frac{1}{N_i(a)} \sum_{x \in Pop, x_i = a} f(x) \tag{3.20}$$

where a is the value of the allele for a certain position of the chromosome, and N is the number of individuals of the population that have the allele. A value called the excess of the allelic value is calculated by:

$$E_i(a) = A_i(a) - \overline{f} \tag{3.21}$$

and the genetic excess is given by:

$$E(A) = \sum_{i=1}^{l} E_i(a) \tag{3.22}$$

The genetic value of a chromosome, which is the expected evaluation of its alleles, is given by:

$$A(x) = E(A) + \overline{f} \tag{3.23}$$

The value chosen as a ameasure of epistasy of an individual is the difference between the evaluation of the chromosome and its genetic value

$$\varepsilon(x) = f(x) - A(x) \tag{3.24}$$

The value chosen to quantify the degree of epistasy of a population is called variance of epistasy (although it takes into account information of two distinct sets). It is given by:

$$\sigma_\varepsilon^2 = \frac{1}{n} \sum_{x \in Pop} [f(x) - A(x)]^2 \tag{3.25}$$

The following is an example for the binary function f(x) = x:

f(00) = 0 f(09) = 1 f(10) = 2 f(11) = 3 $f_{av} = 1.5$

Table 3.2 Value of epistasy with binary representation

Value of the allele	$A_i(a)$	$E_i(a)$
*0	[(f(00)+f(10))/2]=(0+3)/2=1.5	1.5 – 1.5 = – 0
*1	[(f(01)+f(11))/2]=(1+2)/2=1.5	1.5 – 1.5 = 0
0*	[(f(00)+f(01))/2]=(0+1)/2=0.5	0.5 – 1.5 = – 1
1*	[(f(10)+f(11))/2]=(3+2)/2=2.5	2.5 – 1.5 = 1

Chromosome (x)	E(x)	A(x)	ε(x)
00	–1+(–0.5)= –1.5	–1.5+1.5=0	0–0=0
01	–1+0.5= –0.5	–0.5+1.5=1	1–1=0
10	1+(–0.5)=0.5	0.5+1.5=2	2–2=0
11	1+0.5=1.5	1.5+1.5=3	3–3=0

In this example, the variance of epistasy results in $\sigma\varepsilon^2 = (0 + 0 + 0 + 0) / 4 = 0$

An example of the Gray Code for the function f(x) = x is shown below:
F(00) = 0 f(01) = 1 f(11) = 2 f(10) = 3 $f_{av} = 1.5$

Table 3.3 Value of epistasy with Gray Code

Value of the allele	$A_i(a)$	$E_i(a)$
*0	[(f(00)+f(10))/2]=(0+2)/2=1	1 – 1.5 = – 0.5
*1	[(f(01)+f(11))/2]=(1+3)/2=2	2 – 1.5 = 0.5
0*	[(f(00)+f(01))/2]=(0+1)/2=0.5	0.5 – 1.5 = – 1
1*	[(f(10)+f(11))/2]=(2+3)/2=2.5	2.5 – 1.5 = 1

Chromosome (x)	E(x)	A(x)	ε(x)
00	–1+0= –1	–1+1.5=0.5	0–0.5= –0.5
01	–1+0= –1	–1+1.5=0.5	1–0.5=0.5
10	1+0=1	1+1.5=2.5	2–2.5= –0.5
11	1+0=1	1+1.5=2.5	3–2.5=0.5

In this example, the variance of epistasy results in

$$\sigma\varepsilon^2 = (0.25 + 0.25 + 0.25 + 0.25) / 4 = 0.25$$

Davidor's study has shown some limitations:

- There is a very high amount of calculations necessary to determine the measure
- The proposed measurement requires a normalization process if a comparison between functions is to be made
- It is also necessary to develop a precise measurement in order to quantify the epistasy over an amount of the population

Nevertheless, it provides information that helps to understand the GA's difficulty to solve a problem and also it allows important observations: the measure of epistasy over the whole space of solutions is different from the measure over a sample of the population, and the measure of epistasy of a function varies with the chromosome representation.

Other Aspects

Other aspects that cause difficulty to GAs are multimodality and noise. These factors, however, are not necessarily dissociated from those mentioned earlier.

Multimodality can cause difficulty for the use of GAs, as the existence of many local optimal points offers the possibility that attractors can deviate the convergence from the optimal point. One example of multimodal functions is the f6.

The function f6 is an inverted version of the function F6 of Schaffer, Caruana, Eschelman, and Das,[28] and is calculated as

$$f(x,y) = 0.5 + \frac{\left(\sin\sqrt{x^2+y^2}\right)^2 - 0.5}{\left(1.0 + 0.001\left(x^2+y^2\right)\right)^2} \tag{3.26}$$

The values of x and y are obtained through a treatment routine over the chromosome of 44 bits. The 22 first positions are used to determine x, and the other 22 are used to determine y. The binary values of these positions are changed to the decimal values and then multiplied by a specific factor, in order to vary from 0 to 200. These values are then subtracted from 100, with x and y finally varying from –100 to 100.

This is an interesting function because it shows many oscillations, having a high number of optimal points close to each other in the $\Re^3$ and only one global optimal point on (x,y) = (0,0). As there is always a minimal point between two neighbor optimal points, this problem offers high difficulty, especially for "greedy" algorithms. The GAs have shown a much better performance, although it is still a difficult optimization.

Noise can make difficult the convergence of GAs. There are different types of noise: noisy representations, noisy functions, or noisy information.[29]

The noisy representation occurs when it is impossible to represent the desired object in an exact way. For example, it is impossible to represent the continuity of the space through a binary representation.

The noisy functions are those that do not determine exactly the desired value, and an approximation is made. An example is the use of series to determine infinite decimal values.

Finally, the noisy information can be due to many circumstances: when the sensors that provide information are limited, the information itself is noisy, the knowledge is uncertain, or even when a human mistake is made.

REFERENCES

[1] Ridley, M., *Evolution*, 2nd ed., Blackwell Science, Cambridge, 1996.

[2] Blickle, T., Theory of Evolutionary Algorithms and Application to System Synthesis, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[3] Koza, J.R., *Genetic Programming II*, MIT Press, Cambridge, 1994.

[4] Goldberg, D.E., Deb, K., and Korb, D., An Investigation of Messy Genetic Algorithms, Technical Report TCGA 90005, TCGA, University of Alabama, 1990.

[5] Smith, S.F., A Learning System Based on Genetic Adaptive Algorithms, Ph.D. thesis, Department of Computer Science, University of Pittsburgh, EUA, 1980.

[6] Harvey, I., The Artificial Evolution of Adaptive Behaviour, Ph.D. thesis, University of Sussex, School of Cognitive and Computing Sciences (COGS), 1993.

[7] Zebulum, R.S., Stoica, A., and Keymeulen, D., *A Flexible Model of a CMOS Field Programmable Transistor Array Targeted for Hardware Evolution*, ICES2000, 274, vol. 1801, LNCS, Springer-Verlag.

[8] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MAs, 1989.

[9] Bishop, O., *Practical Electronic Filters*, Babani Electronics Books, 1991.

[10] Banzhaf, W., Francone, F., and Nordin, P., On some emergent properties of variable size evolutionary algorithms, in http://www.ai.mit.edu/people/unamay/icga-submissions.html, *ICGA'97 Workshop on Evolutionary Computation with Variable Size Representation*, 1997.

[11] Wu, A. and Lindsay, R., Empirical Studies of the Genetic Algorithm with Noncoding Segments, in *Evolutionary Computation,* 3 (2), 121.

[12] Harvey, I. and Thompson, A., Through the labyrinth evolution finds a way: a silicon ridge, in *Proceedings of the First International Conference on Evolvable Systems (ICES96)*, vol. 1259, LNCS, Springer-Verlag, 1997, 406.

[13] Thompson, A., Temperature in natural and artificial systems, in *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, Husbands, P. and Harvey, I., Eds., MIT Press, Cambridge, 1997, 388.

[14] Gero, J.S., Novel models in evolutionary designing, in *Proceedings of Second Asia-Pacific Conference on Simulated Evolution and Learning* (SEAL98), Volume 2, Canberra, Australia, November 1998.

[15] Fonseca, C.M., Fleming, P.J., An overview of evolutionary algorithms, in *Multiobjective Optimization, Evolutionary Computation Journal*, MIT Press, Cambridge, 3, 1, 1995, 1.

[16] Wilson, P.B. and Macleod, M.D., Low implementation cost IIR digital filter design using genetic algorithms, in *IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, Chelmsford, UK, IEE, 1, 1993, 4.

[17] Schaffer, J.D., Multiple objective optimization with vector evaluated genetic algorithms, in *Genetic Algorithms and Their Applications: Proceedings of the Frist International Conference on Genetic Algorithms*, Grefenstette, J.J., Ed., Lawrence Erlbaum, Hillsdale, NJ, 1985, 93.

[18] Forrest, S. and Mitchell, M., What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation, in *Machine Learning 13,* 285.

[19] Horn, J. and Nafpliotis, N., Multiobjective optimization using the niched Pareto genetic algorithm, IlliGAL Report 93005, University of Illinois, Urbana-Champaign, 1993.

[20] Cieniawski, S.E., An investigation of the ability of genetic algorithms to generate the tradeoff curve of a multi-objective groundwater monitoring problem, Master's thesis, University of Illinois, Urbana-Champaign, 1993.

[21] Haykin, S., *Neural Networks – A Comprehensive Foundation,* Mcmillan College Publishing Co., 1998.

[22] Whitley, D., Deception, Dominance and implicit parallelism in genetic search, in *Annals of Mathematics and Artificial Intelligence,* 5, 1992, 49.

[23] Davidor, Y., A naturally occuring niche and species phenomenon: The model and first results, in *Genetic Algorithms: Proceedings of the Fourth International Conference*, Belew, R.K. and Booker, L.B., Eds., Morgan Kaufmann, San Mateo, CA, 1991, 257.

[24] Mitchell, M., Forrest, S., and Holland, J., The royal road for genetic algorithms: fitness landscapes and GA performance, in *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life,* Varela, F. and Bourgine, P., Eds., MIT Press, Cambridge, 1991.

[25] Rowe, J. and East, I., Deception and GA-hardness, in *Proceedings of the 1NWGA,* Vassa, Finland, Alander, J., Ed., 1995, 165.

[26] Mitchell, M., Holland, J., and Forrest, S., When will a genetic algorithm outperform hill climbing?, in *Advances in Neural Information Processing Systems 6,* Cowan, J., Tesauro, G., and Alspector, J., Eds., Morgan Kaufmann, San Mateo, CA, 1994.

[27] Jones, T., A description of Holland's royal road function, in *Evolutionary Computation, 2 (4), 409.*

[28] Schaffer, J.D., Caruana, R., Eschelman, L. J., and Das, R., A study of control parameters affecting online performance of genetic algorithms for function optimization, in *Proceedings of the Third International Conference on Genetic Algorithms*, Shaffer, D., Ed., Morgan Kaufmann, San Mateo, CA, 1989.

[29] Miller, J.F., and Thomson, P., Combinational and sequential logic optimization using Genetic Algorithms, in *Proceedings of the First IEE/ IEEE International Conference on Genetic Algorithms in Engineering Systems* (GALESIAS - 95), UK, 1995, 34.

CHAPTER 4

Evolutionary Computation: A Tool for Computer-Aided Design of Electronic Circuits

This chapter presents applications of evolutionary computation in the optimization of analog and digital integrated circuits. Optimization problems have been a standard area of application for Evolutionary Computation. As a matter of fact, the case studies reported here fall into the domain of a more traditional use of Evolutionary Algorithms. We divide this chapter into analog and digital VLSI chips optimization. In both classes of cases studies, we emphasize the need to satisfy multiple objectives, turning this into a challenging area of research. Furthermore, we also present a case study where, besides performing optimization, the Evolutionary Algorithm also performs the task of topology selection. This is a step towards the utilization of artificial evolution in the search of innovative topologies, a sort of application presented in the following chapters.

The first section of this chapter deals with the optimization of analog integrated circuits. Particularly, we address the problem of CMOS (*Complementary Metal Oxide Semi-conductor*) and BiCMOS (Bipolar + CMOS) operational amplifiers (OpAmp) optimization. Three cases are discussed: a standard CMOS transconductance amplifier; CMOS OpAmp optimization targeting low-power applications; and optimization of BiCMOS OpAmps. The second section discusses the subject of digital VLSI chips optimization, covering the following topics: logic synthesis; technology mapping; and testability.

4.1 OPTIMIZATION OF ANALOG VLSI CHIPS

This section describes case studies in which Evolutionary Algorithms are used to optimize the analog portion of VLSI chips. Although constituting only a small part of the total area of modern chips, analog circuitry is usually the limiting factor of their overall performance. The current trend towards the achievement of low-power, low-area, and high-speed analog cells may increase the complexity of VLSI analog design, if hard specifications have to be met. This constitutes the major motivation for this set of applications.

Particularly, we target the design optimization of CMOS OpAmps (*Operational Amplifiers*) and transconductance amplifiers, since they are the most widely used building blocks in analog electronics. As it will become clear, the design optimization of operational amplifiers can be handled by Evolutionary Algorithms if it is interpreted as a search task. Particularly, the multi-objective nature of this problem, where the area, speed, power consumption, and other important features of the chips should be considered, turns search algorithms into a more adequate approach than mathematical optimization tools.

The reader can refer to Appendix A where a brief discussion on basic concepts of MOS transistors is performed. We start by discussing the problem of OpAmp

design optimization. Following, we describe the use of GAs to tackle this problem, by first defining the *representation*, *genetic operators,* and *fitness evaluation function*. Finally, we present the results for three classes of problems: the design of a standard Miller Operational Transconductance Amplifier (OTA) cell; the design of low-power cells; and the design of BiCMOS transconductance amplifiers. In the case of BiCMOS amplifiers, we extend the use of Evolutionary Algorithms to the selection of the circuit topology, in addition to the cell optimization.

4.1.1 OpAmp Design Optimization

Operational Amplifiers are the most versatile and widely used building blocks in analog electronics. They consist of coupled differential amplifiers, presenting a high voltage gain, high input impedance, and low output impedance. In order to control the gain and passing band, OpAmps often employ external feedback. Nowadays, they are realized in integrated circuits through different technologies, such as bipolar, CMOS, and BiCMOS. This chapter analyzes the evolutionary design of CMOS and BiCMOS OpAmps, and the next chapter describes the evolutionary design of bipolar amplifiers. Whereas in the case of bipolar transistor-based circuits the designer's creativity is usually applied to the conception of different circuit topologies, in the case of CMOS design, the creativity is used to set the transistors' sizes of a particular topology, and, as a consequence, select the transistors' operating regions. The latter is defined as the *cell sizing* or *transistor sizing* problem.

OpAmp design optimization encompasses a large set of specifications or objectives, around 20, which should be considered along the design process.[1] Nevertheless, only the most important specifications are effectively incorporated into the design methodologies, constituting the *design plan*. It is important to decide, beforehand, which specifications should take part in the design plan, as well as the relative importance of each specification. It is common that, when a circuit is optimized to fulfill only one or two specifications, the other characteristics may become nonsense. Human design relies on the solution of a system of equations. However, when many objectives are included in the system, many solutions may exist. It is usually up to the designer to assess which solution fits better according to the relative importance of each objective.

When cell sizing is performed, each point of the design space consists of the transistors' sizes, biasing current, and compensating capacitance of a target circuit topology. By setting the transistors' sizes and biasing currents, the MOS transistors operating regions are determined (see Appendix A). MOS transistors may operate in strong, weak, and moderate inversion. The strong inversion region, in which $V_{GS}–V_{TN}$ is greater than 0.2 Volts, is the most commonly used in analog design. The weak inversion region, in which $V_{GS} – V_{TN}$ is around 100 mV, is characterized by the low power consumption of the device, being usually employed in micro-power applications. Nonetheless, the use of this operating region also results in the following drawbacks: low speed of the device, increase in its area, and reduction of the load driving capacity. Another way to characterize the weak inversion region is

through equation (4.1):

$$I_D \quad < \quad 2 \cdot n\beta U_T{}^2 \tag{4.1}$$

where β is proportional to the transistors' dimensions *W/L*, U_T is the thermal voltage (26mV at 300K), and *n* takes values between 1 and 1.5, according to some transistor manufacturing features. The moderate inversion region is intermediate between the weak and the strong inversion regions.

4.1.2 Problem Representation

In the particular domain of our problem, each chromosome encodes, by means of a particular mapping, an operational amplifier. The representation refers then to the way this mapping is performed. A sized CMOS operational amplifier can be characterized by a list of real numbers representing transistors' sizes, biasing current, and, if it is the case, a compensating capacitance. Each OpAmp feature is represented in a chromosome integer string, so that each string element serves as pointer to the actual value of the OpAmp feature. Since we are using a string representation, we classify our EA as a Genetic Algorithm. This representation is illustrated in Figure 4.1.

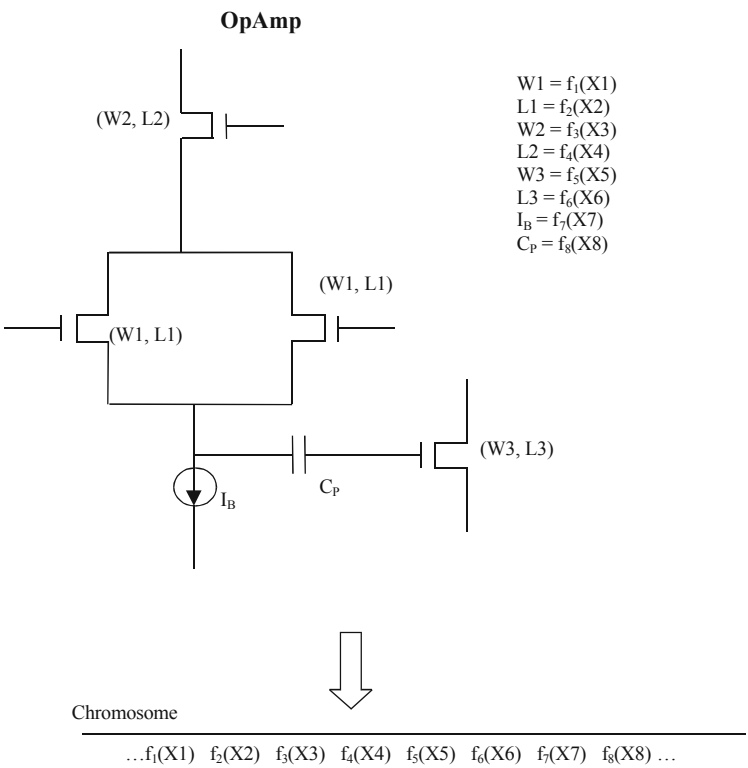


Figure 4.1 Mapping between a CMOS structure and the GA chromosome.

The functions f_i, shown in this figure, perform a simple conversion, whose general expression is given by:

$$f : \mathrm{I} \rightarrow \Re \tag{4.2}$$

$$y = \frac{x}{k_2} + k_1, \qquad x \in (0,1,...,N-1); \qquad y \in [C_{mín}, C_{máx}]$$

In the above expression, y and x are the actual value taken by the OpAmp feature and the value of the associated string position, respectively. While each string position can assume N different integer values (alphabet cardinality), each amplifier feature is constrained to values between C_{min} and C_{max}. These constraints are set according to the referred feature and to the technology being used. For instance, if the feature is a particular transistor width, C_{min} and C_{max} will stand for the minimum width allowed by the technology, $W_{mín}$, and the maximum width chosen by the user, $W_{max,}$, respectively. The constants k_1 and k_2 are determined in order to make the conversion between x and y.

One of the most important aspects in choosing a particular chromosome representation is the amount of previous designer knowledge used in it. Our representation requires the designer to supply minimum and maximum constraints, $C_{mín}$ and $C_{máx}$, for each amplifier feature. Furthermore, in order to avoid meaningless OpAmps, the differential input pairs are also constrained to have the same sizes, as illustrated in Figure 4.1.

4.1.3 Genetic Operators

Chromosomes of fixed length have been used in this application. Therefore, we only need to use the basic GA operators, selection, crossover, and mutation. Particularly, one-point crossover and exponential selection were used.

4.1.4 Fitness Evaluation Function

The multiple objective optimization methodology EMM, introduced in Chapter 3, has been applied to this problem. This methodology is more appropriate to this problem due to the following reasons:

- The use of the EMM makes possible the inclusion of OpAmp performance specifications as a guide to the optimization process. This set of requirements is defined by the user, according to the particular design specifications. We remember that other multiple-objective methodologies for optimization, such as the Pareto-Optimal, do not consider directly the user requirements.

- It can be experimentally shown that the use of simpler approaches, such as fitness aggregation methods with fixed weights, does not produce

satisfactory results. On the one hand, the dynamic nature of the weights typical of the EMM methodology helps the GA to escape local optima.*

Comparative experiments have also been performed to determine a good value for the α parameter used in the EMM methodology (see equation (3.8) in Chapter 3). In these experiments, the combination of a = *0.7* with a value of *c = 0.9* (equation (2.10) in Chapter 2) for the exponential selection method produced better results for this class of application.

4.1.5 Case Studies

This section describes the application of the GA model described previously in the optimization process of three classes of designs: the standard Miller OTA cell; OpAmp topologies targeted to low-power consumption; and a BiCMOS OpAmp topology.[2]

4.1.5.1 Miller CMOS OTA: GAs Rediscovering Human Design

The Miller OTA is a two stage amplifier, whose compensation capacitance introduces the Miller effect, and presents a low output impedance for most of its frequency range. We apply a genetic algorithm together with the multi-objective strategy defined previously to optimize *gain*, *gain-bandwidth product*, *power consumption*, *area,* and *phase margin (*see Appendix A*)*. These performance statistics have been measured through simulation by small signal and operating point analysis. Particularly, the SMASH simulator for electronic circuits has been employed here.

The genetic algorithm manipulates the transistors' sizes, biasing current, and compensating capacitance through the representation shown previously. The values of k_1 and k_2 (equation 4.2) define lower and upper boundaries for the variables manipulated by the genetic algorithm. The transistors' dimensions, *W* and *L*, may take values between 5μm and 105μm with a step of 1μm (using $k_1 = 1$ and $k_2 = 5$). The biasing current, I_B, may take values between 1.5μA and 2.5μA with a step of 0.01μA (using $k_1 = 100$ and $k_2 = 1.5$). Finally, the compensating capacitance C_P may assume values between 1pF and 6pF, with a step of 0.05pF (using $k_1 = 20$ and $k_2 = 1$).

As previously described, the multi-objective evaluation strategy uses a vector of objectives' specifications supplied by the user. For this particular case, the features specification have been set to: minimum of 80dB DC gain; minimum GBW of 2MHz; maximum power consumption of 200μW; maximum area of 5,000μm^2; ** and minimum

* Remember that, in the context of multiple-objective optimization, local optima are defined as those points in which only a sub-set of the objectives is satisfied.

** The OpAmp area has been estimated in a simplified way, just by computing ($\Sigma W \cdot L + 1{,}000C_p$) over all transistors. The second term takes into account the influence of the compensating capacitance in the area.

phase margin of 70°. Table 4.1 compares the features between a human made design and the best design obtained by the genetic algorithm. Figure 4.1 compares the transistor sizes of the human made and the evolved designs. The cell schematic is shown in Figure 4.2.

Table 4.1 Comparison between a human-made Miller OTA circuit and one synthesized by the Genetic Algorithm.

Features	Human Made Design	GA
Design Space	----------	10^{24}
Gain	71.2 dB	72.7 dB
GBW	2 MHz	1.6 MHz
Slew-Rate	3.78 V/µs; – 1.56 V/µs	2.05 V/µs; –1.82 V/µs
I_B	2.50 µA	2.31 µA
C_P	1 pF	1.15 pF
Power	527.8 µW	237.9 µW
Area	1,929 μm^2	3,020 μm^2
Phase Margin	65°	55°
Technology	3 µ n-well	3 µ n-well
R_L	100 k	100 k
C_L	10 pF	10 pF

The results on Table 4.1 show that the GA arrived at an OpAmp with performance statistics close to the specified values, which were deliberately set to hard ones. Comparing to the hand made design, we can observe that they present similar gain and GBW statistics. However, the OpAmp synthesized by the GA features less than half of the dissipation observed in the hand made design, at the expense of a larger area. This is due to the design plan chosen in this particular case. The phase margin achieved by the GA cell is lower than the human made one. If stability is critical in the OpAmp application, the phase margin can be further improved by increasing the Miller capacitance. The slew-rate has not been taken into account as an objective by the Genetic Algorithm, because its measure requires a time consuming transient analysis. Nevertheless, the values obtained (2.05V/µs and –1.82V/µs) compare well with the hand made ones, taking into account that the GA cell dissipates less than the human made counterpart.

It can also be observed from Table 4.1 that the design space is constituted of 10^{24} possible sized OpAmps. Nonetheless, the Genetic Algorithm sampled only 90,000 cells, by running 10 executions with a population of size 30 along 300 generations. This illustrates one of the most important features of a Genetic Algorithm, which is its efficiency in sampling large search spaces.

Figure 4.3 presents the schematic of the Miller Transconductance amplifier. Transistor pairs (T1,T2), (T3,T4), and (T7,T8) have been deliberately constrained to have equal sizes for evolution. Some of human design guidelines are:

- High value of $(W/L)_6$, since transistor T6 provides gain

- High value of $(W/L)_5$, which improves the output swing
- $(W/L)_1 > 1$, since transistor T1 also provides gain

The GA arrived at these design strategies without any kind of previous knowledge being supplied to the system, thus illustrating the GA's potential to rediscover human made design rules.

From Figure 4.2, it can be verified that the GA cell follows closely the human made cell in all respects, except for the lower ratio of $(W/L)_5$.

GA	Human Design
T1 = T2 = (25, 7)	T1 = T2 = (25.2, 15.3)
T3 = T4 = (5, 8)	T3 = T4 = (9.2, 9.56)
T5 = (76, 16)	T5 = (54.2, 4.3)
T6 = (103, 6)	T6 = (114.2, 4.56)
T7 = T8 = (18, 21)	T7 = T8 = (12.2, 9.3)
I_B = 2.31μA	I_B= 2.5 μA
C_P= 1.15p	C_P= 1pF

Figure 4.2 Simulation file for the Miller OTA cells produced by the GA and by a human designer.

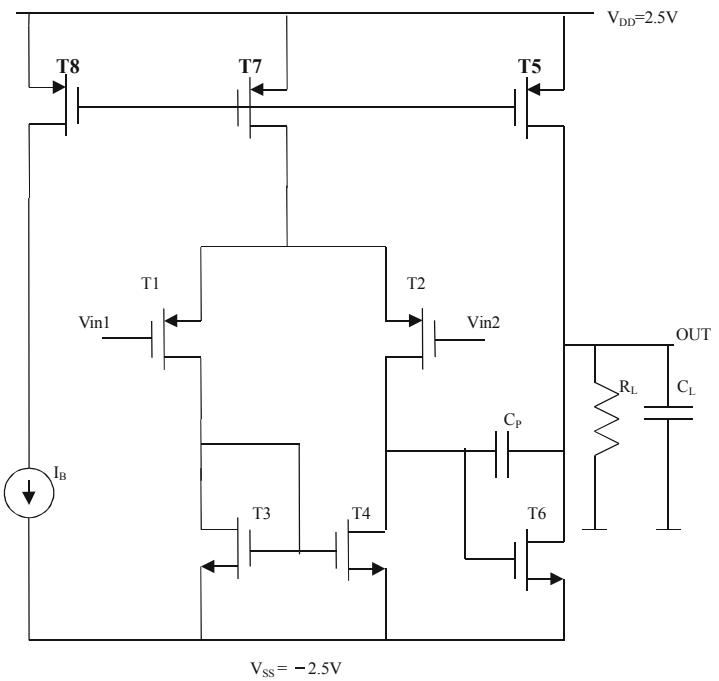


Figure 4.3 Schematic of the Miller Transconductance amplifier.

Let us look at the behavior of the Energy Minimization Methodology (EMM) when applied to this problem. Figure 4.4 presents graphs which depict the average values for the five objectives along the evolutionary process. This average is computed over all individuals and also over all the GA executions. As we can see, there is a clear tendency towards the minimization of the area and to the maximization of the phase margin. The GA also tries to keep the other objectives under the specifications.

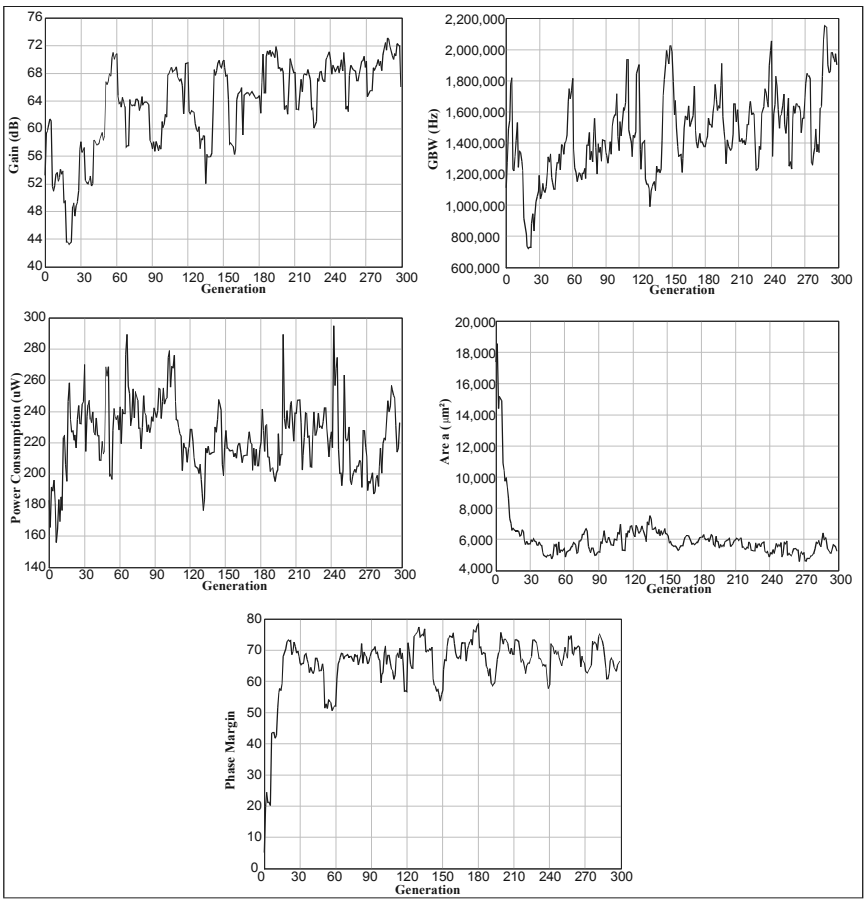


Figure 4.4 Average values of the five objectives considered by the evolutionary process for the optimization of the Miller OTA cell. The graphs show Gain, GBW, Power Consumption, Area, and Phase Margin.

4.1.5.2 Low Power Operational Amplifiers: New Design Strategies

Low power design is a very challenging and active research topic in electronics, since many areas, from mobile communications to space applications, benefit from

very low power consumption circuits. Particularly, the experiments in which we report on low-power design have focused on three OpAmp topologies: class A operational amplifier; OTA with simple output; and OTA with Cascade output.

Referring to the chromosome representation, the transistor dimensions, W and L, may take values between 2µm and 102µm with a step of 1µm (using $k_1 = 1$ and $k_2 = 2$) for the OTAs cells; and values between 3µm and 103µm with a step of 1µm (using $k_1 = 1$ and $k_2 = 3$) for the class A OpAmp. The difference is due to the fact that distinct technologies are used. The biasing current I_B may take values between 0.01µA and 1µA with a step of 0.01µA (using $k_1 = 100$ and $k_2 = 0.01$). Finally, the compensating capacitance of the class A OpAmp, C_P, may assume values between 1pF and 6pF, with a step of 0.05pF (using $k_1 = 20$ and $k_2 = 1$).

The fitness evaluation function has been designed to optimize gain, GBW, DC voltage at the amplifier output, power consumption, area, and phase margin. The design plan, in the format *(Minimum Gain; Minimum GBW; Maximum Output DC voltage; Maximum Power Consumption; Maximum Area; Minimum Phase Margin)* has been set to: (*120 dB; 300 kHz; 0.1 V; 8µW; 7,000µm²; 65°)* for the class A OpAmp; (*80dB, 200 kHz* ; *0.05 V*; *10µW; 5,000µm²* ; *70°*) for the simple OTA; and (*120dB, 200 kHz* ; *0.05 V*; *10µW; 15,000µm²* ; *70°*) for the cascade OTA. Contrasting to the Miller OTA experiments, no external circuitry was used to set the DC output of the amplifier. As a consequence, another objective had to be included, the minimization of the absolute value of the DC output voltage. The level 5 CMOS transistor model, which simulates more accurately the weak inversion region, has been used in these experiments.

Table 4.2 displays the performance statistics of the best amplifiers obtained in these experiments.

Table 4.2 Results on the synthesis of three low-power operational amplifiers: performance statistics for the Class A OpAmp, Simple OTA, and Cascade OTA.

Features	Class A	Simple OTA	Cascade OTA
Design Space	10^{32}	10^{34}	10^{44}
Gain	80 dB	63.6 dB	88.5 dB
GBW	700 kHz	200 kHz	150 kHz
Bias Output	0.73V	1.3 mV	0.85 mV
Offset	–1.71µV	34.6 nV	3.37 nV
Slew-Rate	690mV/µs; –360mV/µs	100mV/µs; –80mV/µs	89mV/µs; –76mV/µs
I_B	0,65 µA	0.10 µA	0.16 µA
C_P	1 pF	----------	----------
Power	20 µW	1.4 µW	8.65 µW
Area	20700 µm²	3,343 µm²	7,189 µm²
Phase Margin	43°	65°	57°
Technology	3 µ Marin	1.2 µ AMS	1.2 µ AMS
C_L	3 pF	3 pF	10 pF

We now analyze separately each of the three results.

Considering the class A OpAmp, the schematic of the best cell achieved by the GA is shown in Figure 4.5. It is important to note that the notion of best circuit is rather qualitative, because there are many objectives being compared. Among the

low-power cells, the class A OpAmp was the most difficult case study to the Genetic Algorithm. This stems from the fact that this amplifier presents an active resistor implemented by transistors T9 and T10, whose objective is to include an extra zero to the OpAmp transfer function. This zero must cancel the second pole of the transfer function. It turns out that the correct dimensioning of transistors T9 and T10 is critical to the stability of the amplifier, characterizing an additional difficulty to the GA.

The evolutionary process involved about 50 GA executions, each one processing 40 individuals along 200 generations, resulting in around 10^6 OpAmp cells being processed.

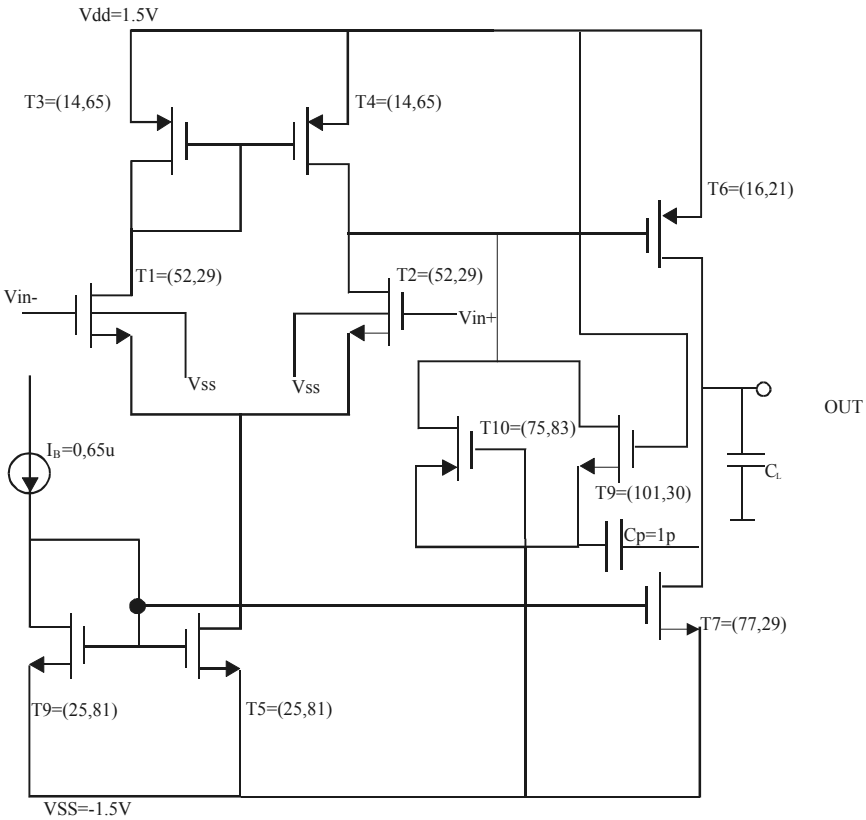


Figure 4.5 Schematic of the class A OpAmp synthesized by the Genetic Algorithm. Transistor sizes are specified by (W,L) in µm.

We can compare this cell with the human made class A OpAmp shown in Salazar,[3] which presents the following statistics: 115 dB gain; GBW of 170 kHz; phase margin of 43.5°; *slew-rate* of 120mV/µs; area around 20,000µm²; power consumption of 3µW; and compensating capacitor of 2 pF. The cell achieved by the GA displayed inferior statistics of gain and power consumption, but this was the means whereby

an acceptable phase margin of 45° was achieved, ensuring the OpAmp stability. The GA cell is also superior in terms of capacitor size,* GBW, and *slew-rate*.

Let us figure out what is going on in this cell. Transistor T1 and T2 have been clearly biased in weak inversion (see Appendix A), a procedure that contributed to increase the value of GBW. The electric current through the first stage of the amplifier (transistor T5) equals 0.65μA, whereas the electric current in the second stage reaches 5μA. This difference is explained by the fact that the ratio $(W/L)_7$ is much bigger than $(W/L)_5$. The high current value of the second stage contributes to the amplifier's dissipation.

The slew-rate of the class A OpAmp is given by the following expression:

$$SR = \frac{I_5}{C_P} = \frac{0.65uA}{1pF} \approx 10^3 mV/us$$

The value of GBW is given by:

$$GBW = \frac{I_5}{2 \cdot n \cdot U_T \cdot C_P}$$

We can observe that the slew-rate has been indirectly optimized through the maximization of GBW. This is a very important fact as the direct measurement of the slew-rate would have required a time consuming transient analysis.

Figure 4.6 depicts the schematic of one the best simple OTA circuits produced by the GA. We can compare its performance with a human designed one found in the thesis of Salazar[3] (1996). We contrast the performance statistics presented in Table 4.2 with the following ones of the human made cell: *65 dB* gain; GBW of *300 kHz;* phase margin of *60°*; *slew -rate* of *120 mV/*μs; area around *5000* μm^2; and dissipation of *2.25* μ*W*. The GA achieved a cell with better statistics of area and power consumption, keeping comparable values to the other specifications. The solution achieved by the GA presented the differential pair transistors, T1 and T2, and the transistor T6 biased in weak inversion. This is a conventional strategy utilized in the case of low-power design. These three transistors determine the gain of the amplifier.

Figure 4.7 compares the transistor sizes for the GA and the human designed OTA cells. From this figure, it can be seen that the GA cell follows closely the human made one in terms of the (W/L) ratios for transistors T1, T2 , T3, T4, and T6. Instead of forcing the current mirror transistors T5 and T10 to have the same sizes, as in the human made design, the GA has deliberately sized these transistors independently. One of the major differences between the two cells is the amount of biasing current used, 0.25μA in the human made design, against only 0.1μA in the evolved cell. As a result, the total drained current and, consequently, the power consumption are smaller for the GA cell. In the case of the total current, its value is 0.26 μA for the GA

* Just like in the case of the Miller OTA, the capacitor size was incorporated to the computation of the cell area which was calculated by: $A = \Sigma (W \cdot L) + 1{,}000 \cdot Cpp$.

cell, against 0.75 μA for the human made cell. This is the most notable feature of the evolved cell.

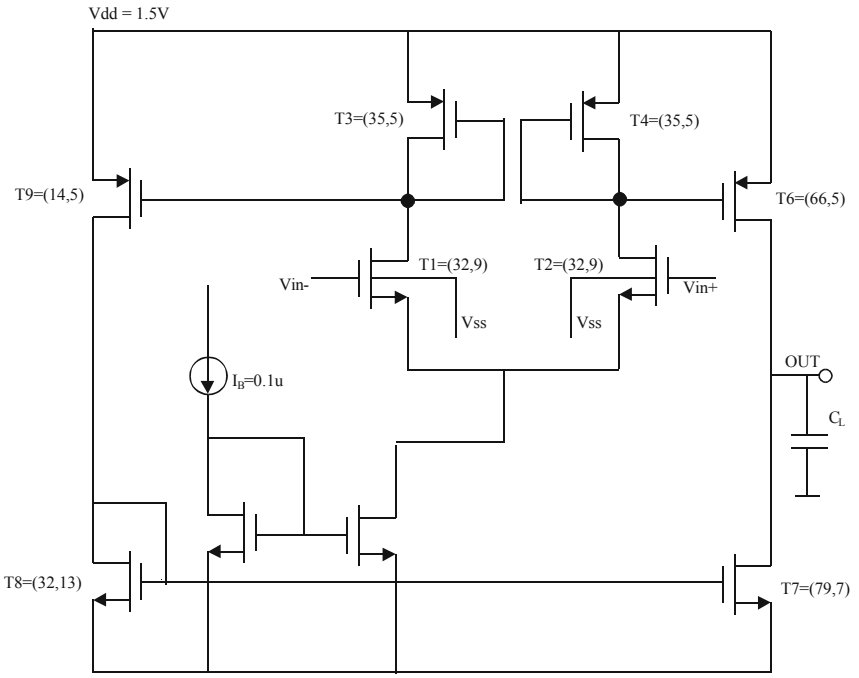


Figure 4.6 Schematics of the simple OTA circuit synthesized by the Genetic Algorithm. Transistor sizes are specified by (W,L) in μm.

GA	Human Made
T1 = T2 = (32,9)	***T1 = T2 = (50,10)***
T3 = T4 = (35,5)	T3 = T4 = (30,10)
T5 = (11,73)	T5 = (40,20)
T6 = (66,5)	***T6 = (60,10)***
T7 = (79,7)	T7 = (20,10)
T8 = (32,13)	T8 = (20,10)
T9 = (14,5)	T9 = (60,10)
T10 = (5,49)	T10 = (40,20)
I_B = 0.1μA	I_B= 0.25 μA

Figure 4.7 Transistor sizes and biasing current of the simple output OTA cells produced by the GA and by a human designer.

We shall now investigate how the cell behaves, by using some mathematical relationships of the simple OTA. The slew-rate (SR) would normally suffer from the low value of the output current. The following expression defines SR:

$$SR = \frac{I_7}{C_L} \cong \frac{10^{-7}}{10^{-12}} = 10^5 V/s = 100 mV/\mu s$$

In the above expression, we used the value measured by the simulator for the OpAmp output current I_7, which stayed around 0.1 μA, close to the value of I_5. Remember that the load capacitance, C_L, is 3 pF (Table 4.2). Therefore, in spite of displaying a low value for the SR, this statistics is compatible with the ones typical of low-power cells. Another concern inherent to the design of low-power OpAmps is the gain-bandwidth product, GBW. This performance statistics is also smaller for low-power amplifiers when comparing to regular ones. Techniques for low-power design attempt to attain acceptable values for GBW within the maximum limit of power consumption. In the particular case of this low-power OTA cell we have:

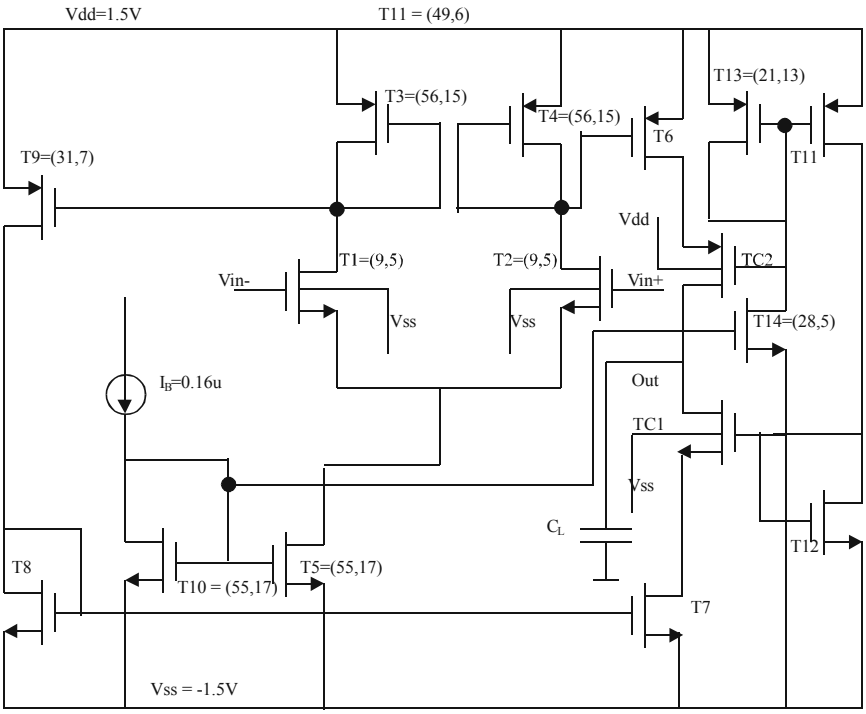
$$GBW = \frac{g_{m1} + g_{m2}}{2} \frac{I_7}{I_5} \frac{1}{C_L} \cong (2.10^{-6}) \cdot (1.5)(0.3 \cdot 10^{12}) \cdot (\frac{1}{2\pi}) \cong 10^5 Hz = 100 kHz$$

From the above expression, it can be seen that there are basically two ways of increasing GBW: increasing the output current I_7, or increasing the transconductances of transistors T1 and T2, g_{m1} and g_{m2} respectively. The latter would hurt the amplifier dissipation; hence, the solution is to increase the transconductances of the differential input pair, which is accomplished by biasing them in the weak inversion region.

The GA was able to find out these rules without any human intervention.

Finally, we now discuss the results achieved for the low-power OpAmp OTA with Cascade output. The objective of the Cascade structure at the output is to improve the circuit's gain. The schematic of one of the best circuits achieved is shown in Figure 4.8.

We can compare this cell with a human designed one, whose performance statistics are given by: 100dB gain; 100kHz GBW; phase margin of 70°; slew-rate of 50 mV/μs; area around 10,000μm^2 ; and dissipation of 5 μW. The GA cell attained better GBW, SR, and area statistics, at the expense of (slightly) inferior statistics of gain, phase margin, and power consumption. We remember again the fact that the slew-rate is indirectly manipulated by the GA, when the same tries to increase the value of GBW. Opposed to low-power conventional strategies, the input differential pair is not biased in weak inversion in the GA cell.



T6 = (98,4); T7=(72,3); T8=(71,15); T12 = (25,13); TC1 = (24,8); TC2=(87,5)

Figure 4.8 Schematic of the Cascade OTA circuit synthesized by the Genetic Algorithm. Transistor sizes are specified by (W,L) in μm.

4.1.5.3 Synthesis of BiCMOS Amplifiers

This application involves two tasks: the circuit topology selection and cell sizing for the single-ended Miller OTA. Four different topologies with two distinct loads are sampled by the Genetic Algorithm:

- The CMOS topology studied in section 4.1.5.1
- A BiCMOS topology with bipolar *pnp* transistors as the differential input pair and *nmos* transistor at the output (*pnp-nmos*)
- A BiCMOS topology with bipolar *pmos* transistors as the differential input pair and bipolar *npn* transistor at the output (*pmos-npn*)
- A BiCMOS topology with bipolar *pnp* transistors as the differential input pair and bipolar *npn* transistor at the output (*pnp-npn*)

Nine different GA executions were implemented. The main difference between this set of experiments and the previous experiments is the fact that the GA has an

additional degree of freedom to choose the amplifier topology. The circuit representation has to be slightly changed: attached to the integer chromosome used to encode the cell sizing, two bits will determine one topology among four possible ones. Therefore, the chromosome now mixes binary and integer representation. The topologies selected by the Genetic Algorithm in the nine executions as the best solution were as follows: four executions selected the *pmos-npn* topology; two executions, the *pnp-npn* topology; two executions, the *pnp-nmos* topology; and one execution selected the CMOS topology. In this experiment, the BiCMOS cells presented better statistics than the CMOS cell. Table 4.3 lists the performance of the best cells for each kind of BiCMOS topology. For each topology, the GA manipulated the MOS transistors' sizes, biasing current I_b, compensating capacitance C_c, and a biasing resistor. Gain, GBW, dissipation, phase margin, and the minimization of C_c have been considered as the objectives in this set of experiments.

Table 4.3 Comparison among the BiCMOS cells achieved by the GA in the experiments involving cell sizing and topology selection for the Miller OTA.

Features	***pnp-nmos***	***pmos-npn***	***pnp-npn***
Gain	64.2dB	71.6dB	84dB
GBW	850 kHz	800 kHz	800 kHz
Dissipation	32µW	36.5µW	55.2µW
I_b	1.89µA	1.52µA	1.5µA
R_b	350k	920k	630k
C_c	2.8 pF	1.2pF	4.25pF
Phase Margin	85°	67°	80°
R_L	100k	100k	100k
C_L	10pF	10pF	10pF

Some conclusions can be drawn from these results. The *pnp-npn* configurations present the best gain statistics, achieving good values for GBW and phase margin. However, the Genetic Algorithm arrived at a high value for the compensating capacitance, 4.25pF, in order to keep a good phase margin. Additionally, the dissipation was too high, a consequence of the use of the *npn* transistors at the output stage. The *pnp-nmos* topology presents a lower gain, but better values of dissipation and compensating capacitance, when compared to the *pnp-npn* topology. The *pmos-npn* topology presented better performance compared to the other ones: it presented acceptable values for gain, GBW, dissipation, and phase margin, with the lowest value for the compensating capacitance. This topology is shown in Figure 4.9.

Theoretically, the *pmos-npn* topology is considered the best of the three BiCMOS topologies.[1] This topology does not face the problem of a bad pnp input stage, which limits GBW. Additionally, the pair Q3-Q5 at the amplifier output produces a larger value of g_m, which is desirable to improve the gain. Generally speaking, it is theoretically expected that the *pmos-npn* topology realize the best GBW with acceptable value for the compensating capacitance. The GA was able to rediscover this kind of knowledge without human intervention.

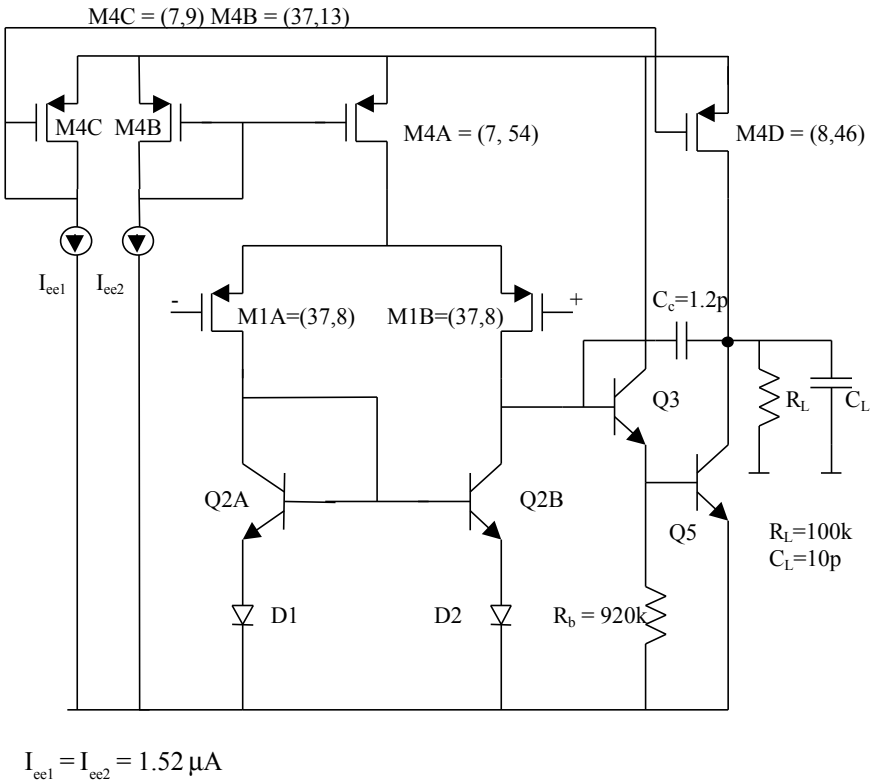


Figure 4.9 Miller OTA in BiCMOS technology: *pmos-npn* topology.

It can be also be observed that the Miller OTA depicted in Figure 4.9 dissipates much less than the one obtained in section 4.1.5.1. This is due to the fact that a value of 20µW was specified as target dissipation in this experiment, against 200 µW in the experiment reported in section 4.1.5.1.

4.2 DIGITAL VLSI DESIGN AND LAYOUT OPTIMIZATION

The use of Genetic Algorithms in the area of VLSI design and layout optimization had been investigated more intensively, at least until the middle 90's, when compared to other areas of . The main reason is that, in this case, the use of GAs is straightforward and driven solely for optimization purposes. No design innovation arises from this class of applications. The GA is used as an optimization tool, and not as a search tool that samples a genome space of novel designs, as it will be described in the next chapters. However, this is an important area of application from the point of view of reducing the costs (area) of integrated circuit manufacturing, mainly in the area of digital chips.

This is an already mature area of research, and the reader can find complete

references on this subject.[4] Nonetheless, since this is also part of the world, this section reviews some of these applications.

Computer Aided Design (CAD) of electronic circuits can be characterized by the following features given by Esbensen and Dreschler:[5]

- Global, multi-modal, and multi-objective
- NP-hard problems
- Mutually dependent problems
- Highly constrained problems

In order to overcome these problems, researchers in the area followed the strategy of artificially dividing this complex and multi-objective problem into sub-problems, and, more importantly, using powerful heuristics together with the GA. In this particular domain of applications, researchers focus on being competitive with the state of the art in CAD design, and, to achieve this purpose, they prefer to make an extensive use of problem-specific knowledge than to use pure Evolutionary Algorithms. From the point of view of research in GAs, this class of applications is not as challenging as in the case of analog circuit optimization. We observed that, in the case of analog circuits, transistor size optimization affects both layout aspects (total area) and also the circuit functionality (gain, bandwidth) of OpAmps. This is not the usual case for digital circuits: in this set of applications the circuit functionality is not affected by the Genetic Algorithm.

In this application area, GAs have been able to deliver competitive solutions in terms of quality, but, so far, they are not competitive in terms of runtime.

This field of research involves applications encompassing most of the implementation issues surrounding digital integrated circuits, starting from the *logic synthesis*, passing through the *technology mapping and physical design* (placement and routing), and finishing at the *testing* phase. We will then describe in this section these three levels of optimization.

4.2.1 Logic Synthesis

Two approaches have been used for logic synthesis, *Fixed Polarity Reed Muller (FPRM)*[6] and binary decision trees minimization.[7] Although this problem is nominated as logic synthesis, the role of the GA is secondary, being used to optimize some aspect of an employed heuristic.

FPRM are products of Boolean variables connected by the exclusive-OR (XOR) operator. Each logic variable can appear either complemented or not-complemented in this product, but never in both forms. The state of the logic variable, complemented or not, is called polarity.[8] The particular choice of polarity for each variable will determine the final size of the expression, thereby influencing the final area of the VLSI chip implementation. An extreme example of this fact is given by the following function *f*:

$$f : B^n \rightarrow B \therefore f = \bar{x}_1 \bar{x}_2 ... \bar{x}_n \tag{4.3}$$

As observed from the above expression, all the variables appear in a complemented form. If we were to express the same function in such a way that all the variables appear uncomplemented, we would have the following expression:

$$f = 1 \otimes x_1 \otimes x_2 \otimes ... \otimes x_1 x_2 ... x_n \tag{4.4}$$

where $\otimes$ is the exclusive-OR operator This expression encompasses 2^n product terms, against only one of the first expression!

The objective of applying GAs to this problem is to find the optimal polarity for each variable. A binary representation is used and each chromosome bit determines the polarity state of one particular variable. The fitness evaluation function is the total number of terms in the final expression, which must be minimized by the GA.[9] Naturally, there is a heuristic that synthesizes the final expression once the polarity of all logic variables is given.

Genetic Algorithms have also been applied to the optimization of Binary Decision Diagrams, called BDDs in the literature.[10] They are acyclic graphs often used to simplify logic functions. GAs are used either to find an optimal sorting for the logic variables or to choose an optimal set of don't-care assignment.[5] Figure 4.10 illustrates the mapping of a simple logic function onto a BDD.

$$F = x1 \cdot x2 + x3$$

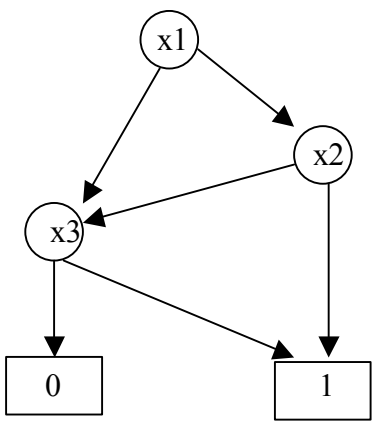


Figure 4.10 Mapping the logic function *F* onto a BDD.

Again, it is important to mention that there is an embedded heuristic that synthesizes the BDD for each particular variable sorting given by the GA chromosome. Due to the secondary role of GAs in these applications, the execution time stays in the order of seconds.

4.2.2 Technology Mapping and Physical Design

This level corresponds to the physical implementation of the logic designed in the first step. As previously explained, the physical implementation and the logic synthesis are treated as independent problems, a procedure that simplifies the overall design process. Nevertheless, it is generally believed that technological trends, such as the decreasing size of transistors, will invalidate this approach in the future, and physical design and synthesis will have to be treated as a single and very complex problem.[5]

We selected the following implementation topics to present in this section: floorplanning; routing; and transistor size optimization.

Floorplanning

This problem can be stated as a Rectangle Packing (RP) problem, where the objective is to place and orient *n* given rectangles such that no rectangles overlap and the area of the smallest rectangle enclosing all *n* rectangles is minimized.[5] This is illustrated in Figure 4.11. For practical purposes, each rectangle represents an autonomous block of digital circuitry.

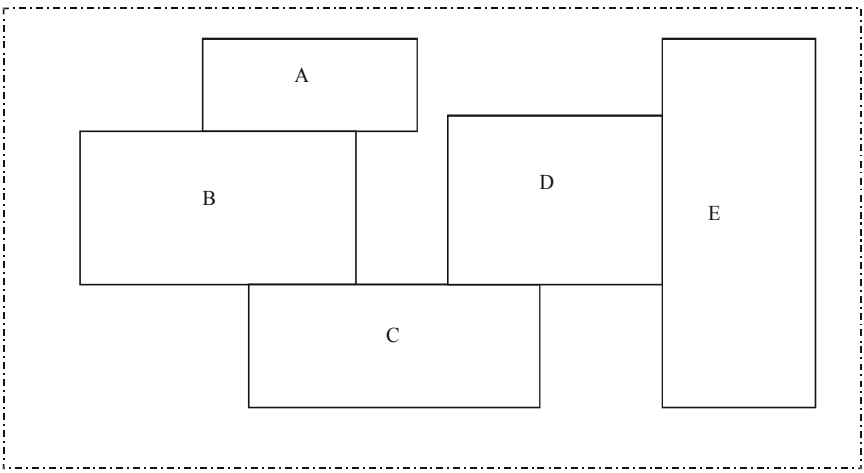


Figure 4.11 Hypothetical Rectangle Packing problem. The five rectangles, A, B, C, D, and E, may represent digital circuitry blocks in the context of digital chips optimization. (Extracted from Esbensen[5] et al.)

Many different representations have been proposed to tackle the RP problem. We will briefly describe two of them, the *coordinate representation* and the *slicing structure*. The coordinate representation is a straightforward one to this problem: *2n* floating values specify the lower left corners of the rectangles, and *n* bits specify their orientation. This representation does not guarantee against unfeasible

solutions, instead, a penalty term is applied to overlapping rectangles. Another particularity of this representation is that, as floating point values are used, the search space size is (theoretically!) infinite. On the other hand, the slicing structure representation encodes a particular placement through a slicing tree. A slicing tree is a Polish expression that recursively creates horizontal and vertical cutlines, defining rooms that contain a single block each.[5] Figure 4.12 illustrates an example of this representation.

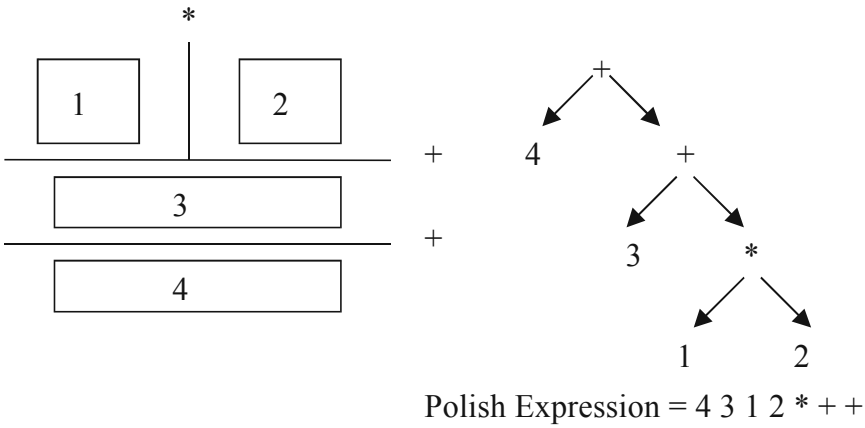


Figure 4.12 Hypothetical example of the slicing structure mapping for the RP problem. The "+" node inserts a horizontal line and the "*" inserts a vertical line. (Extracted from Esbensen[5] et at.)

When using the Slicing Structure encoding, a particular placement will be mapped to a Polish expression of length *2n – 1*. Contrasting to the coordinate representation, all genetic operators will preserve the solution feasibility and there is no need to use penalty factors.

Departing from a representation used to solve the Rectangular Packing task, the floorplanning problem can then be addressed. The task is summarized[11] in the following way:

> *A floorplanning problem is specified by the following:*
>
> *1) A set of blocks, each of which has k ≥ 1 alternative implementations. Each implementation is rectangular and either fixed or flexible. For a fixed implementation, the dimensions and exact pin locations are known. For a flexible implementation, the area is given but the aspect ratio and exact pin locations are unknown.*
>
> *2) A specification of all nets and a set of paths. Capacitance of sink pins, driver resistances of source pins, internal block delays and capacity and resistance of the interconnect are also*

needed to calculate path delays;
3) Technology information such as the number of routing layers

Each output solution is a specification of the following:

1) A selected implementation for each block.
2) For each selected flexible implementation i, its dimensions w_i *and* h_i*, such that* $w_i h_i = A_i$ *and* $l_i \le h_i/w_i \le u_i$*, where* A_i *is the given area of implementation i and* l_i *and* u_i *are given bounds on the aspect ratio of i, which is assumed to be continuous.*
3) An absolute position of each block so that no pair of blocks is closer than a specified minimum distance. Since multi-layer designs are considered, it is assumed that a significant part of the routing is performed on top of the blocks.
4) An orientation and reflection of each block. The term orientation of a block refers to a possible 90 degree rotation, while reflection refers to the possibility of mirroring the block around a horizontal and/or a vertical axis.

Therefore, we can verify that, in addition to the task of rectangle packing, floorplanning also encompasses the selection of an implementation for each block or rectangle. In the case of flexible implementations, the rectangle dimensions w_i and h_i will also compose the solution. Information on the orientation and reflection of each block should also be provided in the solution. Esbensen and Kuh[11] present the following chromosome structure for the floorplanning problem with *b* blocks:

1) A string of b integers specifying the selected implementation of all blocks. The i'th integer identifies the implementation selected for the i'th block.
2) A string of real values specifying aspect ratios of selected flexible implementations. The i'th value specifies the aspect ratio of the i'th selected flexible implementation.
3) An inverse Polish representation of length 2b – 1 over the alphabet {0,1,...,b–1, +,}. The operands 0,1,...,b–1 denote block identities and +,* are operators. The expression uniquely specifies a slicing-tree for the floorplan, with + and * denoting a horizontal and vertical slices, respectively.*
4) A bitstring of length 2b representing the reflection of all blocks. The reflection of the i'th block is specified by bits 2i and 2i+1.
5) A string of integers specifying a critical sink for each net, used when routing the nets. The i'th integer identifies the critical sink for the i'th net.

The third item of the above representation stands for the slicing structure

representation for the rectangle packing problem. Items 1, 2, and 4 refer to the implementation of each block. Finally, item 5 refers to the routing information, also included in this representation. The routing problem will be described next.

The decoder to the above chromosome is very specialized. Given a slicing structure, a specific algorithm is applied, assuring that a minimum layout area structure for that Polish expression is achieved. After a layout compaction, the area of the smallest rectangle enclosing all blocks is computed, providing then a fitness measure of the chromosome.

Routing

The objective of an automatic routing tool is to connect the cell pins while meeting any layout constraints. Typical constraints are to minimize the overall area, avoid meandering paths and thereby reduce delays, and minimize lengths of critical nets. This is therefore a highly multi-objective task. Interconnection wires are assumed to have widths that meet the design rules imposed by the technology used and to be adequately separated from adjacent wires.[4] VLSI routing algorithms operate in a two phase mode: first a global routing is performed determining the topologies of nets and their relative position; second, a detailed routing phase is performed, where the final interconnection specifications among the module terminals are determined in terms of layers, vias, and track assignments.

Due to the high density and complexity of modern chips, routing should be broken into several smaller problems, which is the function of the global router.

It is preferable to blend routing and placement, instead of tackling them as independent problems. Placement and routing are in fact dependent problems. As seen before, the placement algorithm finds position to the macro-cells or blocks that can fit into a minimal size rectangle, thus optimizing area. However, this same placement may increase the area taken by the routing nets: if the floorplaning algorithm puts two blocks that are connected by a large bus far away from each other, the chip area will increase due to routing.

The macro-cells or blocks are not always obstacles to global routing. We can have *feedthroughs,*[4] which is a routing area that passes through a particular cell or block. They exist when there is unused space within a cell, or when the routing wire consists of a metal layer not used in the particular cell. Feedthroughs contribute to reduce the chip area.

Global routing is carried out by constructing a global routing graph, whose edges correspond to the routing regions and vertices correspond to the intersection of the routing regions. To compute a global route for a specific net, vertices representing the terminal are added at appropriate locations. Finding the global route is equivalent to finding a minimum-cost subtree in the routing graph that spans all the terminal vertices. This tree finding can be done using standard shortest-path, spanning-tree, or Steiner-tree algorithms. The Steiner problem is NP-complete, and exact algorithms have an exponential worst case time complexity. Genetic based algorithms have then been applied to the Steiner global routing problem. Details are outside the scope of this book; readers can refer to References 12 and 4.

Transistor Size Optimization

Transistor sizing does not affect the functionality of digital circuits as deeply as in the case of analog circuits, described previously in this chapter. Nevertheless, transistor size optimization can reduce the area and improve speed of the logic cells in digital integrated circuits. In the case of transistor sizing of OpAmps, it has been verified that many objectives should be considered, i.e., gain, bandwidth, slew-rate, area, power dissipation, and phase margin. In the case of logic gates, the fitness evaluation function should consider only power dissipation, speed (delay of the logic gate), and area. Rogenmoser et al.[13] have applied GAs to optimize digital circuits, particularly inverters, NAND gates, and flip-flops. The fitness evaluation function considered the delay time and the power dissipation of the circuits, through a weighted sum of these two objectives. The HSPICE simulator was used in this application. HSPICE simulations are very accurate, but they may be very time consuming for experiments. In this particular application GAs presented similar performance to the one attained by conventional optimization techniques. This is possibly due to the fact that this task is simpler (but not less important) than the analog counterpart.

4.2.3 Testing

Once the chip is fabricated, it must be tested to ensure that it functions as designed. Testing of integrated circuits is an increasingly important problem, which accounts for a significant percent of the total design and production costs of ASICs. What kind of problems can prevent a chip from working correctly? In the particular case of digital ICs based upon logic gates, one common problem is the stuck-at fault.[4] Figure 4.13 exemplifies this problem in the case of a NAND gate. The transistor level implementation of this gate is depicted in the figure. It can be observed that an unwanted resistive short appears between the output and ground. If the resistance is too low, the gate output will be stuck at 0, and the gate will not work.

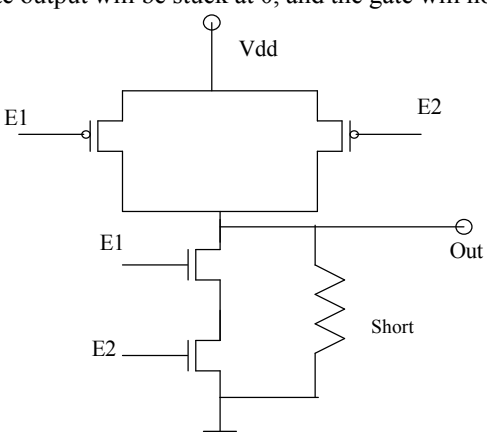


Figure 4.13 Fabrication defect on a NAND gate.

Naturally, when we talk about digital chips with thousands of gates, a large variety of manufacturing defects can happen, contributing to a faulty behavior of the chip. For this reason, a large amount of research efforts have been devoted in the last years to the development of more efficient algorithms for the Automatic Pattern Generation (ATPG) for digital circuits. GAs have also been applied to this area. According to Corno et al.:[14]

> *... The main advantages of GA-based ATPGs when compared to other approaches, like the topological or the symbolic ones, are their lower CPU time requirements, the higher fault coverage they can reach and the capability of dealing with larger circuits (more than 30,000 gates). On the other side, the main drawback consists in the lack of completeness (they can not identify untestable faults) and, more in general, in their difficulty in finding a test sequence for the hard-to-test faults. Moreover, using a GA-based ATPG often requires a careful setting of many parameters which strongly affect its performance, and no method yet has been found, able to automate parameter tuning phase.*

According to the description above, it can be observed that, unlike traditional GA applications in CAD for electronic circuits, genetic based tools outperform conventional tools also in terms of CPU time requirements.

This task can be stated in the following way: a particular defect can only manifest into an observable fault (incorrect chip output) for some particular testing sequences. How to find testing sequences that cover all the possible defects and faults? The general objective of ATPGs is to generate testing sequences with a high fault coverage, making possible the construction of an efficient testing sequence for manufactured chips. Testing sequences are inputs applied to the chip. To each testing sequence, the chip output is compared to the output of one chip that is free of faults. A very simple example clarifies this issue. Figure 4.14 depicts a simple combinational logic with two gates. Suppose that the output of the AND gate is stuck at logic level "0." Which test pattern (input set A, B, … S) will detect this fault? Although the circuit is simple, it is a hard-to-detect fault: only when the inputs A to J are at logic level "1," and the inputs from K to S are at logic level "0," an observable fault will appear. In this case, the output "Z" will be "0"(Y is stuck at "0"), when it should be "1." This particular fault detection can actually be a hard problem for a GA.

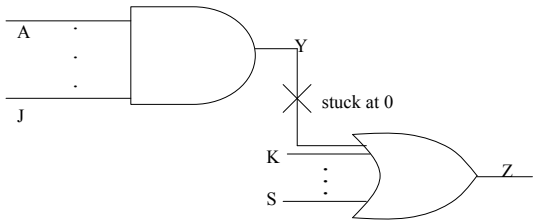


Figure 4.14 Test generation example. (Extracted from Mazumder and Rudnick,[4] 1998.)

The GA representation for this problem is straightforward: a binary string in which each bit represents one input. In the above case each chromosome would be made up of 19 bits, representing the inputs from A to S. The fitness evaluation function is in principle very simple: if the circuit response to the chromosome is correct, the fitness is minimum; if the circuit response is incorrect, the fitness is maximum (the chromosome is able to detect the fault). Nevertheless, this fitness function would not drive the GA to the response in this "pathological" case, since there is only one vector out of 2^{19} that produce the correct answer. The GA would work like a random search, and very unlikely find the solution. We can help the GA by inserting in the fitness evaluation function a term rewarding the number of "1" logic levels at the AND inputs, A to J, and the number of "0" levels at the OR inputs (K to S). In this case, the GA finds the response in less than 7 generations.[4]

Testability is even more important in the case of synchronous sequential circuits. In this particular case, the chromosome represents a sequence of inputs applied to the circuit. One of the most successful GA-based tools for testability of sequential circuits is the GATTO algorithm.[14] This algorithm works in three steps:

- Given a list of faults, one particular is selected at random to serve as target.
- The second phase aims at generating the test sequence for the target fault with a Genetic Algorithm: each individual is a test sequence to be applied starting from the Reset state. The fitness function is the weighted sum of the number of gates and flip-flops having a different value in the good and in the faulty circuit.
- After a test sequence is generated in phase 2, the third phase simulates this circuit to determine whether the sequence can detect other faults in the list.

The distinguishing feature of this algorithm is the fact that the faults are targeted one by one, and not concurrently. The GATTO algorithm was further improved by the use of parallel processing. Different groups of processors are defined and each group will run a GA to solve the same problem in parallel; nonetheless, each group will implement a different fitness evaluation function. The fitness evaluation function for each group of processors differs in their weights. This approach then implements competing subpopulations: each subpopulation is evaluated by slightly different

fitness evaluation functions, but they are driven towards the same goal. This is similar to the procedure described in chapter 3.

Optimization algorithms for CAD of electronic circuits are assessed through benchmarks, many of them publicly available. In the case of ATPG of sequential circuits, one of them is the ISCAS'89 benchmark. Table 4.4 partially shows the results of the improved GATTO for some circuits. The performance is given by the number of faults covered and by the time taken by the algorithm.

Table 4.4 Results of the improved GATTO algorithm to the ISCAS'89 benchmark. Experiment executed on 16 workstations. (Extracted from Reference 14.)

Circuit	Total Faults	Detected Faults	Time (s)
S208	215	150	45
S298	308	273	47
S382	399	366	51
S400	424	380	58

4.3 SUMMARY

Evolutionary Computation has been applied to perform transistor sizing of analog cells. Cell sizing deeply affects the behavior of operational amplifiers in terms of gain, GBW, slew-rate, and other statistics. Moreover, the use of GAs can reduce the chip area and power dissipation.

Transistor sizing of OpAmps is a highly multi-objective problem, and human designers only consider a set of the more important ones. On the other hand, the list of objectives manipulated by the GA, seen in this chapter, can be further increased, by adding intrinsic noise, common mode rejection ratio (CMRR), power supply rejection ratio (PSRR), and many others. As the number of objectives grows, GAs will be more competitive with human techniques. In the very low-power designs, GAs can be particularly useful to derive novel ways of cell sizing to meet the performance criteria. This concept was demonstrated in this chapter in the case of an OTA cell with simple output.

Finally, there may be an engineering problem where the designer may have to choose among different OpAmp topologies to attend a set of objectives. GAs can be an asset in this area as well, since different topologies can be sampled in one evolutionary run. This was demonstrated in the case of the Miller OTA amplifier, where four different topologies were sampled by the GA.

This chapter also reviewed the use of GA as an optimization tool to support layout automation of digital integrated circuits. In addition to the physical layout, GAs have successfully been used in other areas of automatic VLSI digital design, such as logic synthesis and circuit testability.

REFERENCES:

[1] Laker, K.R., and Sansen, W., *Design of Analog Integrated Circuits and Systems*, Mc Graw-Hill, New York, 1994.

[2] Zebulum, R.S., Pacheco, M.A.C., and Vellasco, M.M.B.R., A novel multi-objective optimisation methodology applied to synthesis of CMOS operational amplifiers, in *Journal of Solid-State Devices and Circuits*, Microelectronics Society - SBMICRO, 8, 1, Wilhelmus Van Noije, February 2000, 10.

[3] Salazar, F.A., Projeto de Células CMOS Analógicas de Baixo Consumo a Partir de Transistores Operando em Inversão Fraca, Master Thesis, Electric Engineering Dept., Catholic University, Rio de Janeiro, Brazil, 1996.

[4] Mazumder, P., and Rudnick, E., *Genetic Algorithms for VLSI Design, Layout and Test Automation,* Prentice-Hall, New York*, December 1998.*

[5] Esbensen, H., and Dreschler, R., *Evolutionary algorithms in computer aided design of integrated circuits*, Winter School in Evolvable Algorithms, Course Notes, Denmark Technical University, Janeiro, 1998.

[6] Green, D., *Modern Logic Design*, Electronic Systems Engineering Series, Addison-Wesley, Reading, MA, 1986.

[7] Bryant, R. E., *Graph-based algorithms for Boolean functions manipulation*, *IEEE Transactions on Computers*, Vol. C-35, 8, August 1986, 677.

[8] Esbensen, H., and Dreschler, R., *Evolutionary algorithms in computer aided design of integrated circuits*, Winter School in Evolvable Algorithms, Course Notes, Denmark Technical University, January, 1998.

[9] Drechsler, R., Becker, B., and Gockel, N., A genetic algorithm for minimization of fixed polarity Reed-Muller expressions, in *International Conference on Artificial Neural Networks and Genetic Algorithms*, 1995, 392.

[10] Drechsler, R, Becker, B., and Gockel, N., A genetic algorithm for variable ordering of OBDDs, in *International Workshop on Logic Synthesis*, 1995, P5c.5.55-5.64, 1995.

[11] Esbensen, H., and Kuh, E.S., *EXPLORER: an interactive floorplaner for design space exploration*, European Design Automation Conference, Switzerland, September 1996.

[12] Esbensen, H., *A Macro-cell global router based on two genetic algorithms,* European Design Automation Conference, Grenoble, September 1994, 428.

[13] Rogenmoser, R., Kaeslin, H., and Blickle, T., *Stochastic methods for transis tor size optimization of CMOS VLSI Circuit,* in Parallel Problem Solving from Nature IV, vol. 1141, LNCS, Springer-Verlag, 1996, 849.

[14] Corno, F., Prinetto, P., Rebaudengo, M., and Reorda, M.S., *GATTO: a genetic algorithm for automatic test pattern generation for large synchronous sequential circuits, IEEE Transactions on Computer-Aided Design*, 15, 8, August 1996, 943.

CHAPTER 5

Evolutionary Computation: A Tool for Analog Electronic Circuit Synthesis

This chapter and the following one represent the core of this book. We describe applications that go beyond the traditional scope of systems' optimization in which Evolutionary Computation has been employed. Evolutionary algorithms are now used to handle the overall process of electronic circuit synthesis. Again, we split the case studies in analog and digital electronics. This chapter presents case studies on analog design, and Chapter 6 does the same for digital design. We describe the evolutionary synthesis of passive and active filters; the synthesis of amplifiers based on bipolar transistors, and the synthesis of a Digital to Analog Converter.

5.1 ANALOG CIRCUITS EVOLUTION

As mentioned in the last chapter, analog circuit design is characterized by a higher level of complexity, when compared to digital design, and its implementation relies much more on the experience and intuition of the human designer. One can not always find well-defined rules to accomplish analog design, which turns this class of applications suitable for the use of evolutionary techniques.[1,2] Even though the majority of the modern integrated circuits use digital technology, analog circuitry is still necessary to implement their interface with the outside world, which is analog.[3] Therefore, the complexity of this class of design, together with the relative scarce number of experts, motivates the research in this field.

Techniques for analog design automation have been reported since the seventies. These methods could incorporate heuristics,[4] expert systems,[5] or simulated annealing.[6] They were limited methods, in the sense that they usually restricted their search to well-known circuit topologies. The representational potential of Evolutionary Algorithms, which allows the codification of arbitrary circuit topologies, motivated their application in the area.

This chapter basically analyzes the following problem: given a search space constituted of circuits of different topologies, ranging from the most bizarre ones to ones conventionally utilized by human designers, let the Evolutionary Algorithm sample this search space and try to find a circuit that conforms to a particular design specification. We emphasize that the search space is not only made up of different arrangements of components, but also of identical classes of topologies, with distinct valued components.

The use of evolutionary techniques to analog circuit synthesis is very recent, but many fields of analog design have been analyzed: passive filter synthesis; active filter synthesis; amplifiers; and data converters.

5.1.1 Synthesis of Passive Filters

In the first applications in this area, Evolutionary Algorithms were used to perform the selection of topologies,[7] or the simple determination of components' values for fixed topologies.[8] The research group led by John Koza developed some pioneering applications in this area, where all the steps necessary to the design of passive filters are handled by a Evolutionary Algorithm. These steps are the determination of size and topology of the circuit, and determination of the components' values.

This section starts by introducing some basic aspects of the design of passive filters. We propose an evolutionary methodology based on Genetic Algorithms to handle the design problem, and describe the representation, fitness evaluation function and results, targeting a standard filter specification. Finally, we describe other approaches to passive filters synthesis that employ other evolutionary methods.

5.1.1.1 Passive Filters: Basic Concepts

Passive filters are electronic circuits that do not employ DC power supplies. They often consist of an arrangement of resistors, capacitors, and inductors, together with an AC voltage source. Particularly important is the class of polynomial filters, whose zeros and poles may be directly found from the roots of the transfer function's numerator and denominator.[9] Four main groups compose the class of polynomial filters: Butterworth; *Tchebyscheff*; inverse *Tchebyscheff*; and elliptic filters. Table 5.1 surveys the main features of these four classes.

Table 5.1 – Main features of the four classes of polynomial filters.

Filter	Description
Butterworth	Flat response in the passing band and in the stop band
Tchebyscheff	Ripple in the passing band, flat response in the stop band
Inverse Tchebysceff	Flat response in the passing band, ripple in the stop band
Elliptic	Ripple in the passing and stop band

The Butterworth response is the one that is closer to the ideal frequency response. Figure 5.1 shows a typical filter response. This figure defines the following parameters: f_p is the superior delimiting value of the passing band; f_s is the inferior delimiting value of the stop band; K_p is the maximum attenuation in the passing band; K_s is the minimum attenuation in the stop band; and f_c is the cut-off frequency.

In spite of being closer to the ideal frequency response, Butterworth filters use more components than the other ones. For instance, in order to achieve a low-pass filter with maximum 1dB attenuation in the cutoff frequency of 1000 Hz and 50 dB attenuation at 1100 Hz, a 68th order Butterworth filter would be required, whereas a 17th order Tchebyscheff filter and a 8th order elliptic filter would also comply to this specification.[9]

The main filter characteristics utilized to designate performance are: amplitude of the frequency response, phase of the frequency response, group delay, impulse

response, and response to the step function. The amplitude of the frequency response has been more commonly used to evaluate filter performance in the context of Evolutionary Algorithms.

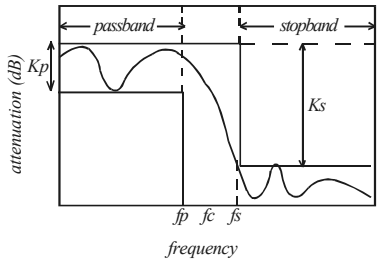


Figure 5.1 Typical frequency response of a low-pass filter (Ver fig. pp. 130 ICES98). (Extracted from Lohn,[3] 1998.)

Though dating from the early sixties, passive filters are still used in medical, video ,and audio applications. For instance, we briefly describe the use of passive filters in the area of music. Passive filters are commonly used to implement the so-called crossover networks. These networks are filters that split incoming signals into high and low frequency bands. In a generic system for musical instruments, after being captured by the microphone and amplified, the signal is reproduced in the loudspeaker. In order to enhance the performance of the system, dedicated drivers are employed for treble (high frequency) and bass (low frequency), also known as tweeters and woofers respectively. The tweeter speaker differs from the bass speaker in its physical constituency: the cone is small and light. These systems are called multiple driver systems.

Figure 5.2 depicts a first order crossover network. The capacitor prevents low-frequencies to reach the tweeter, something that would seriously damage it. The inductor, on the other hand, will prevent high-frequencies to achieve the bass unit, something that would result in interference. Also shown in Figure 5.2 is the frequency response of the crossover network, where we can see that it is simply made up of low and high pass filters. The point where their responses overlap is called crossover frequency.

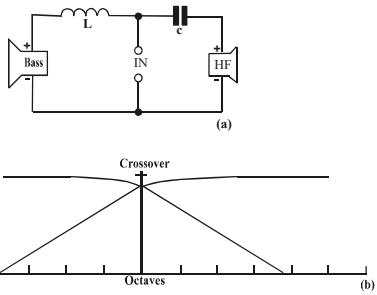


Figure 5.2 First order crossover filter and its frequency response. (Extracted from Bishop.[11])

First order filters presents a 6dB per octave roll-off,[9] which is too gentle to this

particular application. Second and third order crossover circuits, shown in Figure 5.3, are more adequate.

Some aspects of the design of crossover networks may render them very complex, and they may have to be carried by a computer. Some of these factors are: the need for fidelity when reproducing the sound of musical instruments rich in harmonics, such as cello; and the fact that the drivers' impedance varies with the frequency. This section will also describe the evolutionary design of a crossover network.

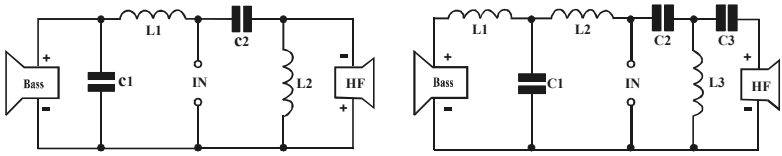


Figure 5.3 Second and third order crossover circuits. (Extracted from Bishop.[11])

5.1.1.2 Representation

Genetic Algorithms have been applied to the synthesis of passive filters. As it has been discussed, the most distinguishing feature of Genetic Algorithms in relation to the other Evolutionary Algorithms is the use of string data structures as the chromosomes. Particularly, we have adopted an integer representation, based on strings of fixed or variable length.

So, the main question is, how do we map a passive filter into an integer string? There are clearly many ways to perform this kind of mapping, some more obvious and direct, others more complex. We choose to present here a very direct mapping, based on the one introduced by Grimbleby[7] (1995). So far, there is no evidence that more complex mappings are more effective than the simpler ones, but we shall discuss more on this later in this chapter.

The main idea of this representation is to divide the chromosomes into genes, similarly to the biological metaphor. Each gene will encode for a particular circuit component, determining its nature, value, and connecting points.

The *nature* of the component is chosen among three possibilities, resistor, capacitor, and inductor. This is consistent with the objective of passive filters evolution. Only one locus of the gene is needed to encode for this feature.

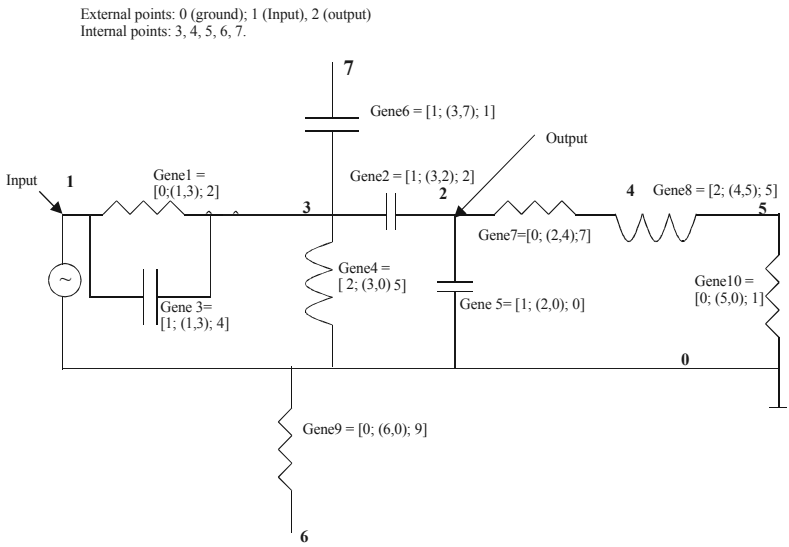
In principle, these components can take an infinite number of real *values*; hence,we could think of a real representation for this feature. However, as we are dealing with a real world problem, we know that we can only count with a finite range of component values. Therefore, we can use an integer representation, where each integer value will point to a particular value a component can assume. It is up to the user to choose these values, and this should be done in a reasonable way, by letting the GA use only commonly manufactured components' values. Normally, one locus will suffice to store this piece of information. The total number of component values will determine the cardinality of the representation alphabet for this particular locus. Note that this cardinality may vary depending upon the component nature.

Finally, the gene must encode the information on where the component terminals are connected. This information allows the GA to encode for an arbitrary circuit

topology in case we do not constrain the encoding. There will be as many loci encoding for the connecting points as there are terminals for the particular component. Each locus will have the cardinality of the representation alphabet equal to the number of circuit's nodes. The GA user must supply the information of how many nodes the evolving circuits are allowed to have.

Still referring to the circuit nodes, they can be classified into two sets, external and internal nodes. The external nodes will be attached to external signals or probing points, such as the input signal, the output node, and ground. In principle, the GA does not select preferentially one kind of node or the other; they are only identified along the chromosome decoding process. It is often advisable, however, to force some loci of the chromosome, relative to the portion of the gene that encodes for the connecting points, to point to external points. This is a way to protect against meaningless circuits, i.e., those without input, output, or ground.

These concepts are summarized in Figure 5.4, where a circuit encoding example is given. Shown in this figure is a hypothetical passive filter topology and its corresponding chromosome, for a particular set of representation choices: a total of eight connecting points, three external and five internal; ten possible resistor values; six possible inductor values; six possible capacitor values. The chromosome, in this example, has a fixed size of 10 genes, hence encoding for circuits of up to ten components. Particular loci of genes 1 and 2 have their values enforced, in order to point to the input and output respectively. Note that genes 6 and 9 did not connect their components' terminals to nodes that connect to the output, and, therefore, are floating. In this case, the simulator usually ignores these components.



Gene = [Component Nature; (Connecting Components); Component Value]

Component Nature: 0 (resistor); 1 (capacitor); 2 (inductor)

Component Value	Resistor (Ohms)	Capacitor (uF)	Inductor (mH)
0	100	0.001	0.01
1	500	0.0005	0.05
2	1000	0.001	0.1
3	5000	0.005	0.5
4	10000	0.01	1
5	20000	0.05	5
6	50000		
7	100000		
8	500000		
9	1000000		

The above values are not preferred ones. Optionally, we can select only preferred manufactured values

Figure 5.4 Mapping between the chromosome and the electronic circuit for a hypothetical passive filter.

Another issue concerning representation is the use of fixed or variable length representation. Variable length representation is more natural to this problem domain, since one does not know, a priori, the number of components the target passive filter may have.

5.1.1.3 Fitness Evaluation

The computation of each individual's fitness is accomplished by performing a circuit simulation in the frequency domain, also called AC analysis. Once the simulation is finished, a file is generated with the frequency response of the particular circuit. The fitness evaluation function is then calculated in the following way:

$$Fitness = -\sum_{i=1}^{n} w_i \cdot |A(i) - T(i)|$$

where n accounts for the total number of samples produced in the simulation; w accounts for a weight vector; and the vectors A and T account, respectively, for the actual and target frequency responses. Therefore, the fitness evaluation function computes an error measure relative to the required frequency response T. In order to emphasize the error measure |A(i) – T(i)|, it is advisable to specify the frequency response in decibels.

Determining the weights of the fitness function is critical to the GA performance. Through much experimentation, the authors arrived at a set of weight values that worked well: a constant value of 600 for the frequency points located inside the passing band; and a constant value of 10 to the other frequency points, i.e., those located at the transition and stop band. The main idea behind these numbers is to penalize more heavily errors in the passing band. Of course, one has to take into account the fact that there is a different number of frequency points in the different bands, and the assignment of the weight vector must also balance the contributions of the bands.

5.1.1.4 Case Study: Low Pass Brick-Wall Filter

We investigate the synthesis of a Brick-Wall low pass filter, a circuit that is a reference to the assessment of new design methodologies.[9,10] The passing band spans from 0 to 1kHz and the stop band starts at 2kHz. The Genetic Algorithm, with the representation and fitness evaluation function previously defined, has been employed. Here follows some additional features of the GA:

- Variable length representation using the UDIP sweeping strategy (See Chapter 3)
- Genotypes consisting of up to 20 active genes
- Each gene may encode for a resistor, a capacitor, or an inductor
- There are seven interconnecting points, four being internal and three being external (ground, input, and output points)
- The GA will choose among 80 different preferred values for each component; those values which are more likely to be commercially available
- The experiment processed a total of 400000 individuals (10 executions, 1000 generations and 40 individuals)

- One-point crossover applied at a rate of 70%
- Mutation rate of 2% per chromosome locus

The circuit of Figure 5.5 displays one of the best responses achieved by the evolutionary process.

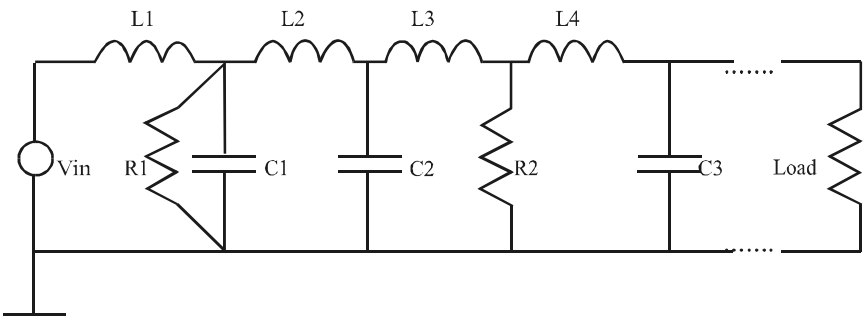


Figure 5.5 Schematic of the best low-pass Brick-Wall filter attained by the evolutionary process.

Except for the load resistor, the above circuit presents nine components. The evolved circuit originally presented 15 components, but six of them were found to be redundant, in the sense that, if taken away from the circuit, the frequency response did not change. Regarding the above figure, the following component values have been obtained by the GA: L1=220μH; R1=1Ω; C1=470μF; L2=470μH; C2=150μF; L3=2.2μH; R2=2.7Ω; L4=22μH; C3=1.2μF. This circuit presents a maximum attenuation of –3dB in the passing band; and an attenuation of –74 dB at the start of the stop band.

Suppose, though, that one is not satisfied with the filter frequency response. The designer may think that an attenuation of –3dB in the passing band is unacceptable. A second experiment has been devised. In this second experiment, the chromosomes encode only the components' values, and the circuit topology is fixed to the one shown in Figure 5.5. More importantly, we adopt the EMM method for multiple-objectives optimization. Three objectives are then considered: maximization of the attenuation from 2kHz onwards; minimization of the negative ripple in the passing band (from 0 to 1kHz); and minimization of the positive ripple in the passing band. The GA presented the following features:

- Fixed length representation as the topology is now fixed
- Genotypes made up of nine genes, encoding for the component values
- As in the first experiment, the GA will choose among 80 different preferred values for each component

- The experiment processed a total of 120,000 individuals (10 executions, 300 generations, and 40 individuals)
- One-point crossover applied at a rate of 70%
- Mutation rate of 2% per chromosome locus

According to the definition of the EMM methodology, the GA performance can be assessed by the value of the system energy (defined in equation 3.1, chapter 3). Figure 5.6 presents the average value of the energy, which, as theoretically expected, decreases along the evolutionary process. The new evolved component values are: L1=10mH; R1=470Ω; C1=3.3μF; L2=47mH; C2=1μF; L3=0.1μH; R2=560Ω; L4=0.22H; C3=0.33μF. To the circuit output, a 500Ω load resistor was connected.

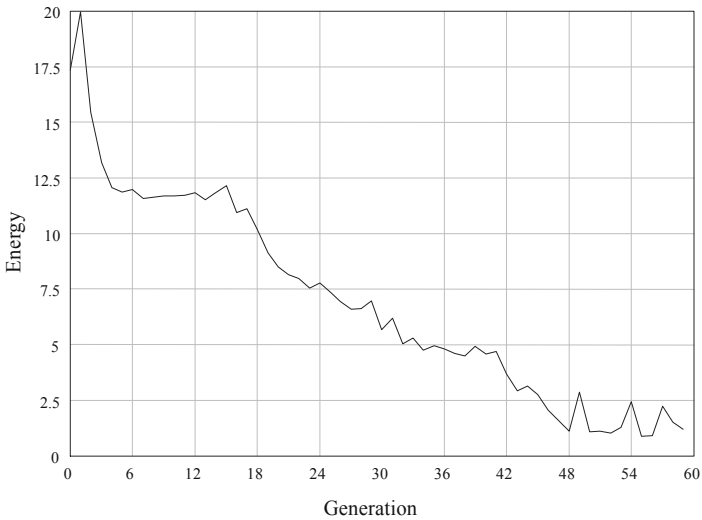
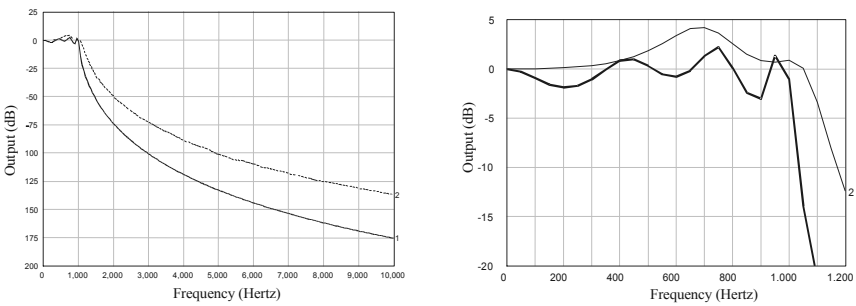


Figure 5.6 Average value of the energy for the second experiment of passive filters' evolution.



(A) (B)

Figure 5.7 Frequency response of the circuits achieved in the first and second phases of the Brick-Wall experiment. Graph (A) presents the frequency responses until 10kHz; Graph (B) focuses on the passing band.

According to graph (B) of Figure 5.7, the first circuit displays both positive and negative ripples, while the second circuit presents only a positive ripple of 4dB.

Table 5.2 depicts a performance comparison among six circuits: the two circuits evolved by the GA, as shown in this section; two Brick-Wall filters evolved through the Genetic Programming algorithm;[10] a conventional *Butterworth* filter; and a conventional *Tchebyscheff* filter. It is clear from this comparison that the GA and GP circuits achieve similar performances in terms of their frequency responses. However, it can be observed that the GA samples less individuals than GP. Furthermore, the GA circuits are slightly more parsimonious than the GP ones. We describe later the details on the use of GP to evolve electronic circuits.

Table 5.2 Comparison of the low-pass filters synthesized by GA, GP, and Butterworth and Tchebyscheff circuits.

	Negative Ripple in the Passing Band	**Attenuation at f = 2kHz**	**Sampled Individuals**	**Number of Components**
GA (1)	-3.04 dB	-73.77 dB	400000	9
GA(2)	0 dB	-49.41 dB	520000	9
GP(1)	-0.53 dB	-62.32 dB	10240000	14
GP(2)	-0.30 dB	-64.67 dB	67840000	28
Butterworth	-3 dB	-60.2 dB	-------	10
Tchebyscheff	-0.1 dB	-70 dB	-------	8

Considering the issue of variable length representation systems, this section focused on the use of the UDIP strategy to evolve the Brick-Wall low-pass filter. The reader should report to chapter 3 to see a survey on the performance of the other sweeping strategies, ILG and OLG, in this problem. These results refer not only to the evolutionary synthesis of low-pass filters, but also to the synthesis of band-pass and notch filters.[11]

5.1.1.5 Other Evolutionary Approaches

This section describes the application of two other evolutionary approaches to the design of passive filters. The first approach refers to the proposal of a developmental representation of electronic circuits embedded in Genetic Programming. This approach has been applied to a wide variety of problems in electronics.[12] The second approach refers to another developmental representation, but, this time, embedded in the context of Genetic Algorithms.[3]

5.1.1.5.1 Genetic Programming

We shall now describe the procedure followed by Koza[13] et al. (1996) to evolve analog circuits through GP, including the representation, evaluation, and some case studies.

Representation

The *representation* is certainly the most important issue on the application of GP to evolve circuits. As studied in the first chapter, GP uses tree data structures as chromosomes. A straightforward mapping between circuit topologies and tree data structures is not obvious whatsoever. As it is stated by Koza[13] et al. (1996):

> *Genetic programming breeds a population of rooted, point-labeled trees (i.e., trees without cycles) with ordered branches. There is a considerable difference between the kind of trees bred by genetic programming and the labeled cyclic graphs encountered in the world of electrical circuits. Electrical circuits are cyclic graphs in which every line belongs to a cycle (i.e., there are no loose wires or dangling components). The lines of a graph that represent a circuit are each labeled. The primary label on each line gives the type of electrical component (e.g., resistor). The secondary label(s), if any, give the component value(s).*

In order to overcome this problem, they came up with a *developmental* approach to perform this mapping. Departing from an initial circuit, called "embryonic," a tree will encode a set of instructions to build a circuit from the initial one. This circuit structure encompasses a fixed and a modifiable part: the fixed part contains the source resistance, output load, source signal, ground, and other essential features of the target circuit; the modifiable part usually consists of pieces of wires that are subject to the application of building instructions encoded in the tree. Figure 5.8 depicts an example of a typical embryonic circuit used in Koza's applications. Apart from the two modifiable wires Z0 and Z1, the remaining parts of the circuit are fixed.

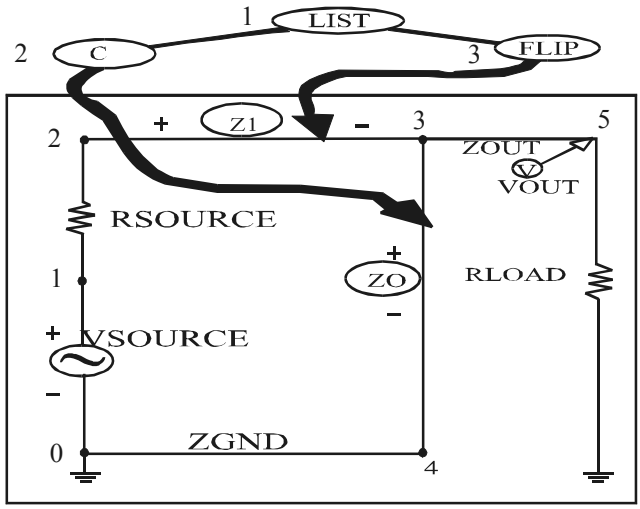


Figure 5.8 Typical case of an embryonic circuit used in GP applications for circuit evolution. (Extracted from Koza[13] et al.)

The embryonic circuit may contain (or not) specific information about the kind of circuit to be evolved. As mentioned in Koza[13] et al. (1996a), the amount of domain knowledge incorporated into the embryonic circuit should be minimal.

So, we need to clarify the means whereby a tree will provide the instructions to change the modifiable wires of the embryonic. As previously described, we can divide the nodes of GP trees into functional nodes and terminal nodes. The function and terminal sets are chosen according to the particularities of the application, and so is the procedure defined here. The functional nodes may belong to two classes: *Component-Creating Functions* and *Connection Modifying Functions*. Component-creating functions point to a modifiable part of the circuit, and create a particular component in this part of the circuit. In the particular case of passive filters, we can have the **R**, **C**, and **L** functions, which create resistors, capacitors, and inductors, respectively. Connection Modifying Functions alter the topology of the modifiable parts of the circuit. Examples of these functions are **Series** and **PSS,** which create, respectively, a series and a parallel connection of a particular component. Both the component-creating functions and the connection- modifying functions will point to a particular wire or component in the modifiable part of the circuit.

Having defined what sort of functions will be present in the GP trees, the terminal set remains to be defined. Why would we need terminals in this kind of mapping? The most obvious reason is that we simply need to end the sequence of instructions encoded by a tree; hence, the **END** terminal is used. Another reason is that, whenever a component is created by the creating functions, a value must be associated to it. This is done by attaching a so-called arithmetical sub-tree to the component-creating functional node. This arithmetic sub-tree will perform arithmetical operations over real numbers, so as to produce a real value. So, we can summarize this discussion

by defining the function ($\Im$) and terminal (Γ) sets:

$$\Im = \Im_{cc} \cup \Im_{cm} \cup \{+, -\}$$
$$\Gamma = \Re \cup \{\text{END}\}$$

where $\Im_{cc}$ stands for component creating functions; $\Im_{cm}$ stands for connection modifying functions; $\{+, -\}$ are the functions used by the arithmetical sub-tree; $\Re$ stands for the real numbers used in the computation of the components' values; and END is a terminal with no specific function other than finish the tree.

Figure 5.9 describes the procedure of circuit development, departing from a hypothetical tree structure. The embryonic circuit is the one shown in Figure 5.8.

We may now ask what advantages and disadvantages this representation brings, particularly if we compare with the more direct representation employed by the GA previously defined. The advantages are the following ones:

- This technique is a solution to the problem of mapping circuits into trees.
- Arbitrary circuits' topologies with different sizes can be mapped through this scheme. These advantages stem from the fact that GP employs a variable length representation.
- It is a very flexible representation, in the sense that different kinds of analog circuits can be represented this way: tests have been done for passive filters, and bipolar transistor amplifiers, but other types of analog circuits, such as CMOS amplifiers, could also be mapped.

This kind of mapping presents the following disadvantages:

- It is not a straightforward mapping, which increases the complexity of the decoder.
- The assignment of components' values is performed in a rather arbitrary way, without concern to the problem of circuit implementation.
- As a result of the fact that GP is being used, methods should be employed to control the size of the evolved circuits.

The first of the above reasons that, at a first glance, would seem unimportant is relevant since one can always argue that the most natural representation to a particular problem ought to be used. Regardless of the system performance, the use of this circuit representation is definitely not natural to this problem. Regarding the assignment of values to the components, we showed that an arithmetic sub-tree was employed to compute the component value. The use of this arithmetic sub-tree

seems a rather arbitrary procedure and, furthermore, commercially non-available values can be generated. However, a simple change in this method would overcome this problem. Finally, as shown in the second chapter, GP presents the drawback of excessive growth in the tree size, and some control methods, such as deleting crossover, parsimonious pressure, and Automatically Defined Functions (ADFs), should be used. As a consequence, we can observe, from the reported results,[12] that GP usually achieves large circuits compared to human designed ones.

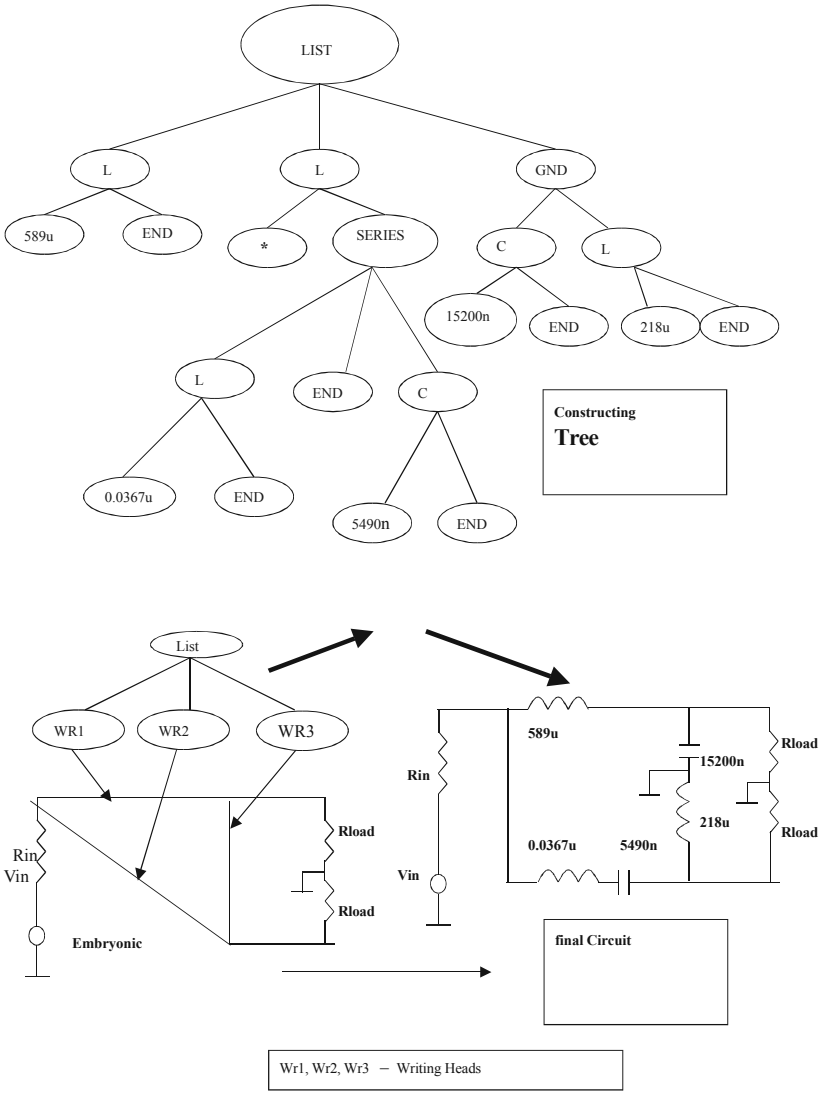


Figure 5.9 Hypothetical example of mapping between the GP tree and an electronic circuit using a circuit-constructing program tree.

Fitness Measure

As expected, the fitness evaluation is accomplished by considering the circuit behavior in the frequency domain. The AC small signal analysis of the SPICE simulator was used to evaluate the circuits' performance. Koza et al. devised the following fitness equation, which has later been used by other authors:[3,14]

$$F(t) = \sum_i [W(d(f_i), f) \cdot d(f_i)]$$

where f_i is the frequency of the fitness case *i*; *d(x)* is the difference between the target and the observed values at frequency *x*; and *W(y,x)* is the weighting for a difference y at frequency x. As explained earlier, the task of setting the weights is the most serious problem in this fitness equation. This is usually accomplished by experimentation. For instance, when GP was used to evolve the Brick-Wall low-pass circuit, the following strategy was used:

> *The fitness does not penalize ideal values; it slightly penalizes every acceptable deviation; and it heavily penalizes every unacceptable deviation.*
>
> *The procedure for each of the 61 points in the 3-decade interval from 1 Hz to 1,000Hz is as follows: If the voltage is between 970 millivolts and 1,000 millivolts, the absolute value of the deviation from 1,000 millivolts is weighted by a factor of 1.0. If the voltage is less than 970 millivolts, the absolute deviation from 1,000 millivolts is weighted by a factor of 10.0. This arrangement reflects the fact that the ideal voltage in the passband is 1.0 volt, the fact that a 30 millivolts shortfall is acceptable, and the fact that a voltage below 970 millivolts is not acceptable.*
>
> *The procedure for each of the 35 points between 2,000 Hz to 100,000 Hz is as follows: If the voltage is between 0 millivolts and 1 millivolt, the absolute value of the deviation from 0 millivolts is weighted by a factor of 1.0. If the voltage is more than 1 millivolt, the absolute value of the deviation from 0 millivolts is weighted by a factor of 10.0. [...] The deviation is overlooked for each of the 5 points in the interval between 1,000 Hz and 2,000Hz (i.e., the don't care band).*

Even though the fitness evaluation method described for the GA was inspired in the evaluation procedure described above, there are some differences. The procedure described in section 5.1.1.3 did not use a don't care band, i.e., the frequency response in the transition band is also considered. Furthermore, the weights' values do not change for different values of the circuit output as described above; they change only according to the particular circuit band.

Case Studies

We show first some evolved Brick-Wall low-pass filters evolved by GP, which served as basis for comparison with the results achieved by the GA. A GP with 320,000 individuals, 89% of crossovers, 10% of reproductions, and 1% of mutations has been used. Figure 5.10, Figure 5.11, and Figure 5.12 show interesting circuits evolved in three different runs, observed at generations 32, 64, and 212, respectively. The circuit of Figure 5.10 presents a seven-rung ladder topology that is very common in *Butterworth* and *Tchebyscheff* circuits. The circuit of Figure 5.11 displays a bridged T arrangement, also employed in human design. While these two designs exemplify the potential of GP to rediscover filter topologies, the fully compliant circuit shown in Figure 5.12 is a novel topology achieved by GP. The frequency response of this circuit is shown in Figure 5.13.

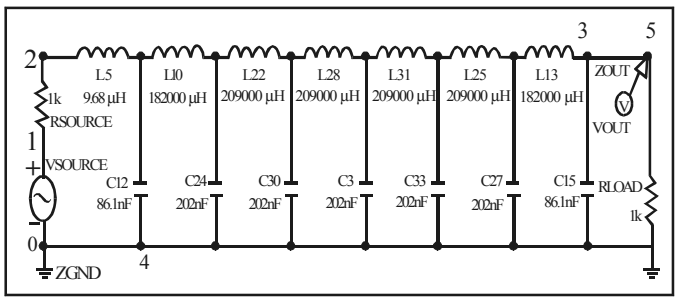


Figure 5.10 Seven-rung ladder Brick-Wall circuit evolved using GP. (Extracted from Koza[13] et al.)

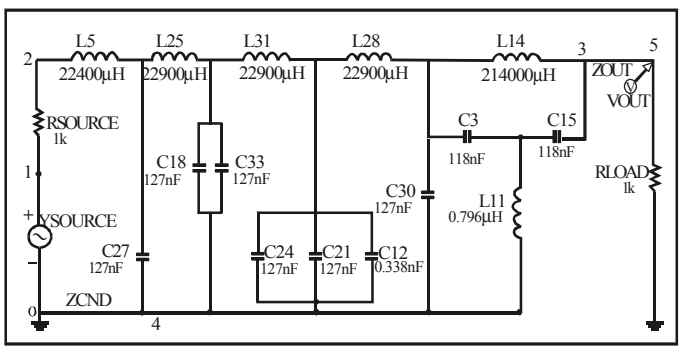


Figure 5.11 Bridged T Brick-Wall circuit evolved by GP. (Extracted from Koza[13] et al.)

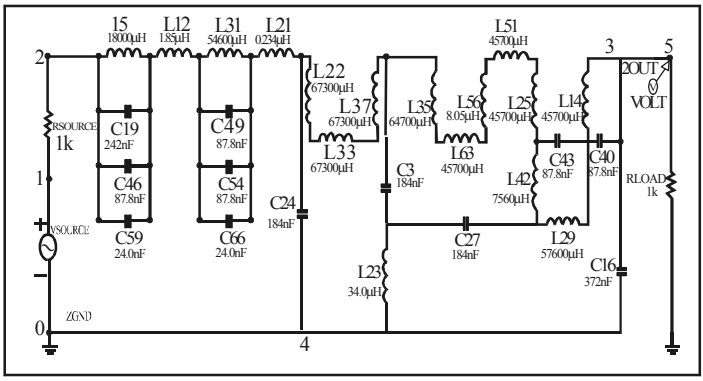


Figure 5.12 Novel topology for a Brick-Wall filter, 100% compliant, evolved by GP. (Extracted from Koza[13] et al.)

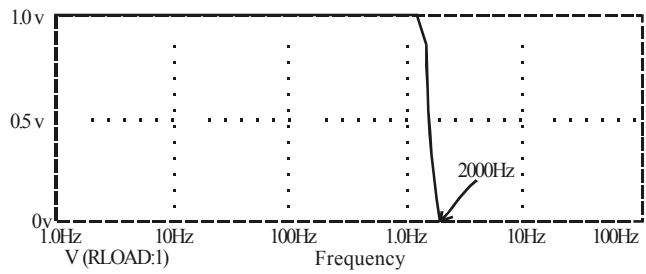


Figure 5.13 Frequency response of the circuit presented in Figure 5.10 – Seven-rung ladder Brick-Wall circuit evolved using GP (Extracted from Koza[13] et al.).

A crossover filter has also been evolved through this methodology. As previously described, the crossover filter is used to separate low and high frequency components in multiple driver systems. The crossover frequency was set to 2,512 Hz, and two outputs must now be probed, associated to the low (woofer) and high (tweeter) frequencies. Apart from this, the procedure is the same as the one described for the Brick-Wall filter. The circuit shown in Figure 5.14 was the best achieved by GP. Both the low-pass and high-pass parts of the circuit are composed by Butterworth topologies.

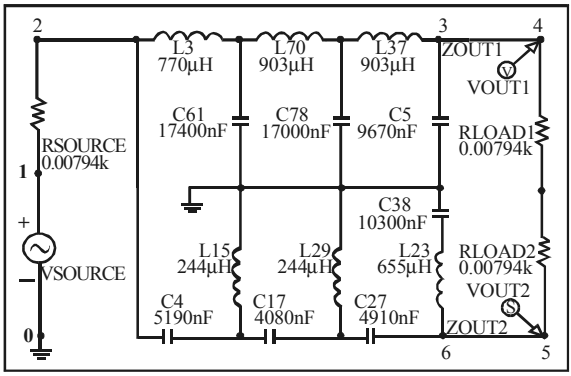


Figure 5.14 Crossover circuit evolved by the GP methodology.

5.1.1.5.2 Genetic Algorithm with Developmental Representation

In section 5.1.1.2 we studied the use of a direct representation of electronic circuits in the context of Genetic Algorithms. The GA model which is presented here employs a developmental representation of electronic circuits, similar to the procedure followed in the case of the GP applications. We start describing the representation, and then the fitness evaluation function and case studies.

Representation

Lohn[3] et al. (1998) selected what they called a linear representation to evolve circuits. According to this scheme, the building unit of the chromosome is a set of bytecodes that carries the instruction of how to insert a particular component in the circuit. As described by Lohn:

> *A fixed number of bytecodes represent each component as follows: the first is the opcode and the next three represent the component value.*
>
> *Using three bytes allows the component values to take on one of 256^3 values, a sufficiently fine-grained resolution. ...*
>
> *The opcode is an instruction to execute during circuit construction. In the current design of our system, we use only "component placement" opcodes which accomplishes placement of resistors, capacitors and inductors.*

As described above, each gene is made up of a fixed number of bytecodes and provides all the necessary information of one component of the circuit. Three bytecodes encode the component value, and one of them, the opcode, encodes both the nature and the component connecting points. The opcode will give an instruction to insert the particular component in the circuit. As in the case of the GP circuit constructing trees, an initial circuit structure, where the chromosome will

develop the circuit, is also used. This initial circuit is very simple, containing an AC voltage source, a source resistance, and a load resistance, as shown in Figure 5.15.

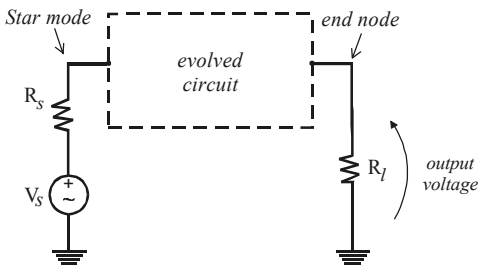


Figure 5.15 Initial circuit in the developmental representation. (Extracted from Lohn[3] et al.)

So, we now explain the circuit development process. Lohn et al. devised five different instructions that are explained in Table 5.3.

Table 5.3 Summary of opcode types used in the developmental representation proposed by Lohn[3] et al. (1998). CN is the current node, and x denotes a resistor, capacitor, or inductor.

Opcode	**Destination Node**	**CN Register**
x-move-to-new	newly created node	assigned the newly created node
x-cast-to-previous	previous node	unchanged
x-cast-to-ground	ground node	unchanged
x-cast-to-input	input node	unchanged
x-cast-to-output	output node	unchanged

Before explaining the meaning of each instruction, we must explain the idea of current node register, introduced in the above table. The current node is the one in which the component will be inserted by the current instruction. After inserting the node, the instruction may change the current node, or leave it unchanged, as shown in Table 5.3. As the chromosome opcodes are interpreted, the current node value is updated as well. The start node, displayed in Figure 5.15, is the first current node, when the developmental process starts. The actions generated by each instruction are concisely defined in Lohn[3] et al. (1998):

> *The unfolding process then proceeds to interpret each opcode and associated component values, updating the CN register if necessary. The x-move-to-new opcode places one end of component x at the current node (specified by the CN register) and the other end at a newly-created node. The CN register is then assigned the value of the newly created node. The x-cast-to- opcodes place one end of component x at the current node and the other at either the ground, input, output, or previously-created node. After executing these opcodes, the CN register remains unchanged....*

> *The list of bytecodes is a variable length list (the length is evolved by the GA). Thus, circuits of various sizes are constructed. When the decoding process reaches the last component to place in the circuit, we arbitrarily choose to have the last node (value in CN) connected to the output terminal by a wire. By doing so, we eliminate unconnected branches.*

Therefore, considering what has been explained above, the developmental process is carried out through a sequential read of the opcodes of a chromosome. It is also said that each chromosome may be constituted of a variable number of opcodes, resulting then in circuits of various sizes. However, the authors of this technique did not provide further details on how they manipulated variable length chromosomes. Finally, we remark that, at the end of the developmental process, the last node is connected to the output node; hence, the output will always be connected to the circuit.

Let us now analyze the advantages and disadvantages of this representation. The main advantages are:

- It is a simple representation, if compared to the GP circuit constructing trees: there are less primitives, resulting in a reduced time to decode the chromosomes.

- Whereas traditional hand-designed topologies, such as ladder constructs, can be realized, this mapping is very unlikely to generate topologies which can not be simulated.

This method brings, nonetheless, the following limitations:

- It is not fully motivated: there is no obvious reason one should prefer this developmental representation against the direct GA representation introduced earlier.

- Due to the small set of instructions, there are classes of topologies that can not be represented using this method. For instance, circuit branches off the "main constructing thread" are limited to a maximum of one node.

- The process of selecting the component values from the three bytes used to encode this information is not detailed in Lohn et al. (1998). Even though care was taken to allow the components' values to be restricted to a reasonable range, these components do not take, in principle, preferred values.

Fitness Evaluation Function

The fitness evaluation function used was similar to the one employed in GP and GA. The absolute values of the difference between the individual's output and the target output are calculated and summed, in the frequency domain. This results in a scalar measure of error between the target specification and the achieved response. Details on the setting of the weights that multiplied the errors were not reported.

Case Studies:

The authors who conceived this representation reported results for two filters: one of them has a practical use in electronic stethoscopes; the other one is a theoretical case, consisting of taking, as specification, the frequency response of a 3rd order Butterworth filter.

The stethoscope filter is intended to filter out high-frequency sounds picked up by a microphone, which makes it difficult to listen to the low-frequency bodily sounds. The cutoff frequency of this filter is 796 Hz. The frequency response exhibited by this circuit is not very difficult to attain, because the transition band is not very sharp. Figure 5.16 shows the schematics of a fully compliant circuit evolved by the GA with developmental representation. Figure 5.17 displays the frequency response of the evolved circuit.

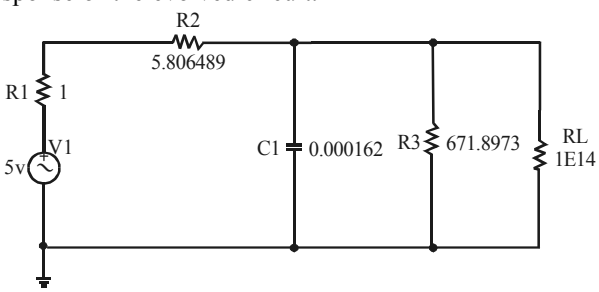


Figure 5.16 Evolved low-pass filter for use in an electronic stethoscope. (Extracted from Lohn[3] et al., 1998)

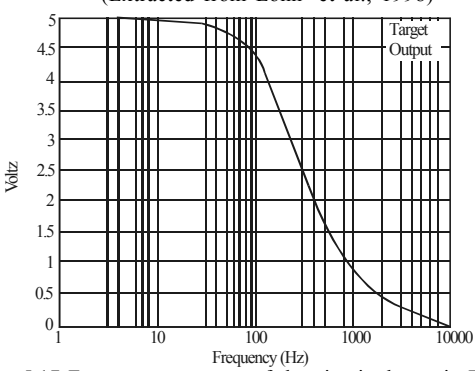


Figure 5.17 Frequency response of the circuit shown in Figure 5.16. (Extracted from Lohn[3] et al.)

The second test was the synthesis of a 3rd order Butterworth filter. The specification can be given by the parameters defined previously in this chapter:

f_p=925Hz K_p=3.0101 dB
f_s=3200Hz K_s=22dB

Figure 5.18 depicts the fully compliant evolved circuit, and Figure 5.19 displays its frequency response.

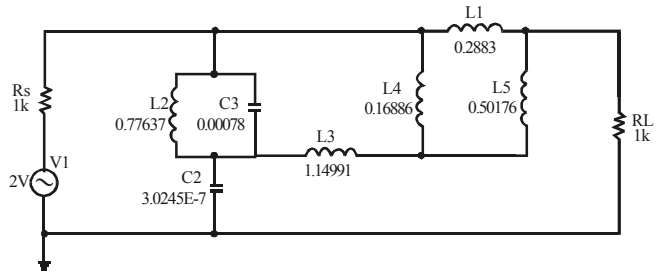


Figure 5.18 Evolved 3rd order Butterworth filter specification. (Extracted from Lohn[3] et al.)

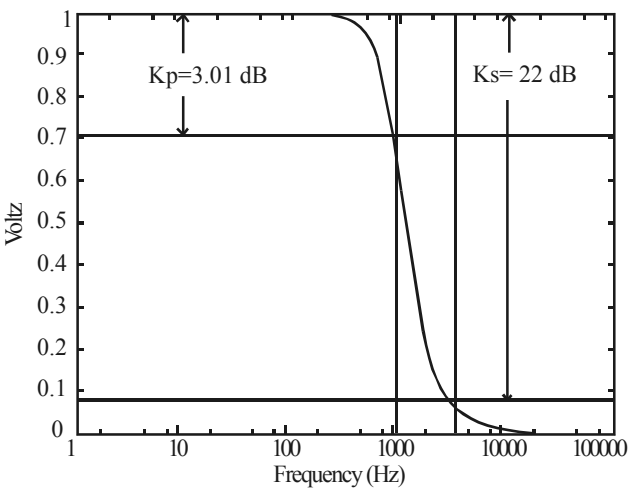


Figure 5.19 Frequency response of the circuit shown in Figure 5.18. (Extracted from Lohn[3] et al.)

5.1.2 Synthesis of Active Filters

Contrasting to passive filters, active filters employ a DC power supply. They are usually made up of basic amplifying devices, such as bipolar or MOS transistors, or

by higher level building blocks, such as operational amplifiers. Particularly, the evolutionary design of an active filter based on bipolar transistors is the main focus of this section. Even though Metal Oxide Semiconductor (MOS) transistors dominate the microelectronics industry nowadays, bipolar transistors are still popular in high-speed applications. As previously mentioned in this book, the BiCMOS technology, which combines bipolar and MOS transistors in the same microcircuit, is a very promising tendency in the electronics industry.[1]

The synthesis of an amplifier for the AM band is described in the following subsections. This circuit works as a band-pass filter, amplifying incoming signals located in the AM frequency band. Particularly, many practical radio amplifiers are still implemented through bipolar transistor technology.[15] We address some important requirements for automatic design tools, such as their ability to process *multiple-objectives*; *speed* to produce a design from scratch; and *implementability* of the produced circuits.

Following we describe the problem and present the main features of the GA utilized, representation, and evaluation. Afterwards, some results are presented.

5.1.2.1 Problem Description

In this example we analyze the particular problem of designing an amplifier for a radio receiver tuned to the AM frequency band. This active filter must amplify incoming signals inside the frequency band ranging from 0.15 to 1.6MHz,[15] whilst attenuating signals outside this frequency band.

Figure 5.20 depicts a schematic of the desired system:

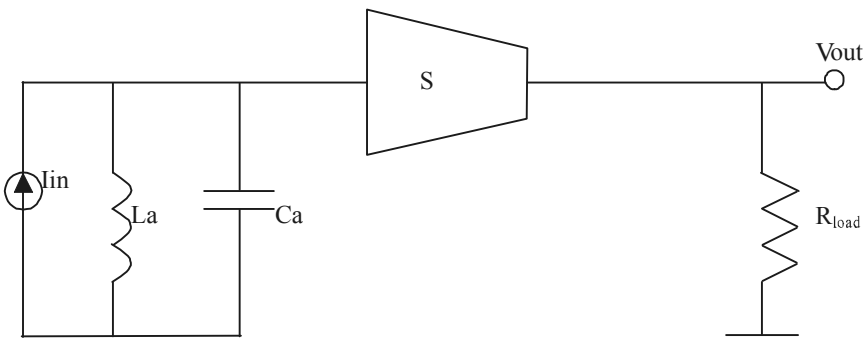


Figure 5.20 Basic schematic of the AM receiver with inductive source.

In Figure 5.20, L_a represents the antenna inductance, and C_a is a parasitic capacitance, assuming values of 3.5mH and 7pF respectively.[19] The input signal, I_{in}, presents a magnitude of 1.5μA (around –117dB). The combination of L_a and C_a works as a trap to the incoming signal I_{in}. This trap has a resonant frequency at 1MHz. S represents the amplifier to be evolved, which must be tuned to the AM frequency band defined above. Finally, R_{load} is the load output to be driven.

Conventional design of active filters often uses operational amplifiers as circuit building blocks. In contrast, this work does not enforce this conventional design principle, and low-level building blocks, such as transistors, resistors, and capacitors, are utilized. Although this procedure increases the design complexity, novel circuits are more likely to be achieved.

5.1.2.2 Representation

We employ a GA using the direct representation described in the passive filter application. As it was described previously, an integer representation based on a linear string has been employed. However, in this case, fixed length chromosomes were used. Furthermore, we now allow the representation of bipolar transistors, due to the particularities of the problem at hand. Figure 5.21 depicts an example of this kind of genotype-phenotype mapping for a common emitter amplifier.

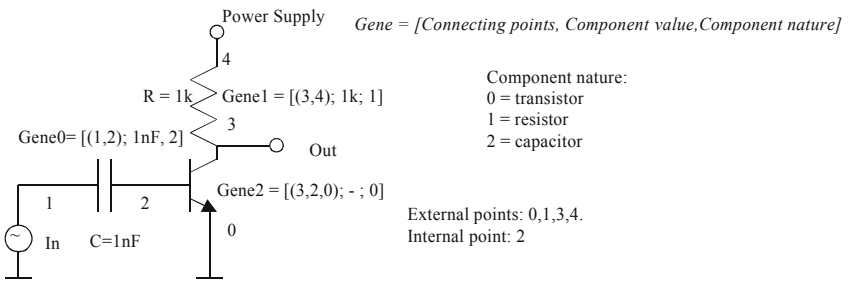


Figure 5.21 Analog circuit representation.

Just like in the case of the passive filters' mapping, the genotypes are made up of genes, each of which encodes a particular component. Each gene determines the nature, value, and connecting points of the related component. We remember that the total number of connecting points is a parameter to be set in this representation. This parameter is critical to the efficiency of the representation: if too few connecting points are considered, the number of possible topologies sampled by the evolutionary algorithm will be limited; conversely, if too many connecting points are considered, a higher number of unsimulatable topologies (with floating components) will arise. Additionally, each connecting point may be classified as internal or external. While the former does not serve for any special purpose, the latter is connected to one of the following signals: power supply, ground, input signal, or probed output (Figure 5.21).

5.1.2.3 Evaluation

The main challenge of applying genetic algorithms or any other search technique to analog design is the multi-objective nature of this task. We saw in the third chapter of this book that multi-objective optimization concerns the need to integrate

vectorial performance measures with the inherently scalar way in which most optimization techniques reward performance. The EMM methodology, which was described in section 3.3.2, is employed in this case study as well.

5.1.2.4 Case Studies

We present the results of three classes of experiments, all of them targeting the synthesis of the AM receiver introduced previously. The first experiment processes a single objective; the second and third experiments handle four and five objectives respectively.

5.1.2.4.1 Single Objective Experiments

This first class of experiments took only the desired frequency response into account. The fitness is given by the following equation:

$$Fitness = \sum_{i=1}^{n} w_i \cdot (V_{out}(i) - V_{in}(i)) \quad \textbf{(5.1)}$$

where V_{out} is the circuit output and V_{in} is the incoming signal at the input of the circuit *S* shown in Figure 5.20. As these voltages are measured in decibels, $(V_{out}(i) - V_{in}(i))$ is the amplifier gain at a particular frequency *i*. *n* is the total number of output samples. This fitness equation is, therefore, akin to the one employed in the passive filters' experiment. The weights w_i take positive values for frequency points inside the AM band, and negative values outside this band. The weights' values have been set through experimentation, assuming a value of +33 for frequency points between 150 kHz and 2 MHz (passing band): –4 for frequency points above 1 MHz; and –1 for frequency points below 150 kHz.

A chromosome made up of 12 genes has been employed in this set of experiments. The components' nature are chosen from four options: npn transistor; pnp transistor; resistor; and capacitor. Eight connecting points are available for the topology arrangement, four of them being external ones (input, output, power supply ,and ground). A power supply of 3V, which is typical of radio batteries, has been used. The evolutionary algorithm can choose among eight different values for resistors and capacitors respectively. The resistor values range from 75Ω to 500kΩ and the capacitor values range from 0.1nF to 100μF. The authors' strategy was to let a small number of different component values be available to the GA, and keep the design space in a manageable size. As a consequence, an interactive involvement with an expert may be necessary to further improve the evolved circuits.

The size of the search space can be calculated from the above values. There are a total of 4 x 8^3 different genes in this representation.* As each chromosome is

* Number of genes = (# different components) · (# connecting points)^2 (# different component values) = $4 \cdot 8^2 \cdot 8$

constituted by 12 genes, one can conclude that the size of the search space is given by $(4 \times 8^3)^{12}$ possible solutions, which is around 10^{33}.

In order to sample this search space, we ran 10 executions of the GA, each one including 40 individuals and 100 generations. Each execution lasted around 30 minutes in a Sun Ultra Enterprise 2 server with one 300 MHz Ultra Sparc processor. The small signal analysis of the SPICE simulator has been used in this set of the experiments. It has been verified that most executions produced circuits that conformed well to the specification. Figure 5.22 depicts the schematic of one circuit achieved in this set of experiments; Figure 5.23 shows its frequency response.

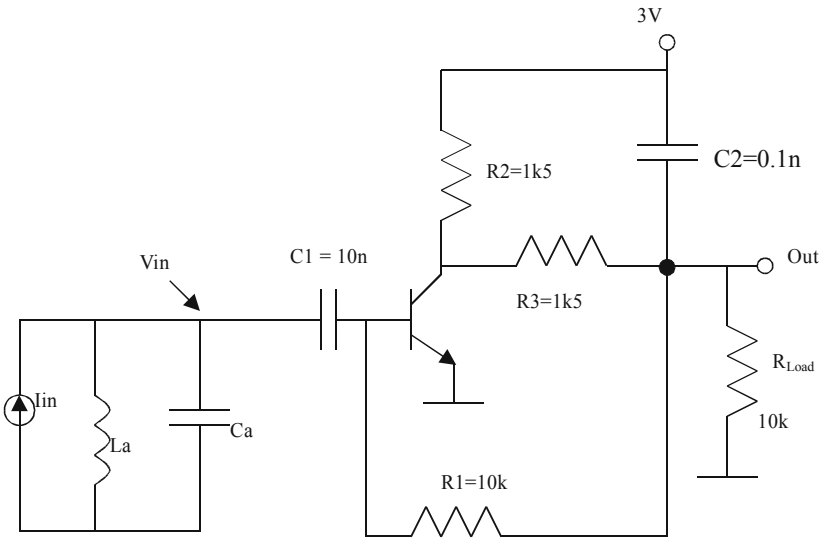


Figure 5.22 Schematic of an amplifier obtained in the first set of experiments.

From Figure 5.22, it can be seen that the evolved solution uses only 6 components (not considering L_a, C_a, and R_{Load}); the other 6 components encoded in the chromosome were not effectively contributing to the circuit behavior. The most interesting aspect of this circuit is its parsimony, which stems from the fact that only one objective had to be fulfilled by the GA. It is also interesting to note that this amplifier is configured in a conventional common-emitter topology. The circuit connected in its collector, R2, R3, and C2, works as a low-pass filter, with cut-off frequency around 5 MHz; the capacitor C1 attenuates low frequency signals. The resistor R1 sets the DC operating point of the amplifier. Even though temperature variations were not taken into account in the fitness evaluation function, this biasing configuration is advantageous to compensate effects of temperature changes.[16]

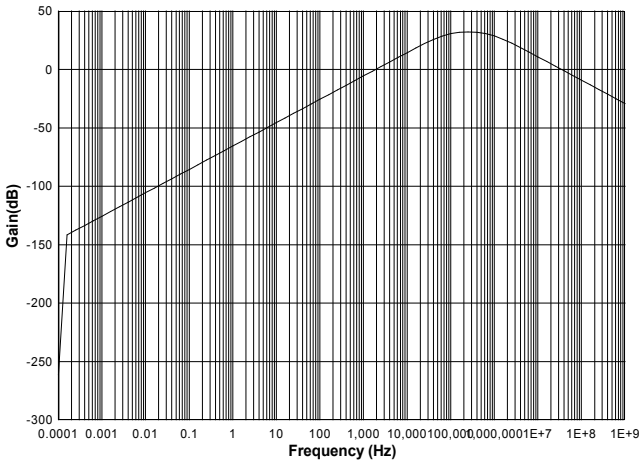


Figure 5.23 Frequency response of the filter shown in Figure 5.22.

The graph of Figure 5.23 shows the amplifier's gain in the frequency domain. Focusing on the AM frequency band, the minimum gain achieved by the circuit is 28 dB, at 1.5MHz; and the maximum gain is 34dB, at 400kHz. Conventional circuits present an average minimum gain of 16dB and an average maximum gain of 37dB.[16]

Further design improvements can be accomplished by optimizing the resistor and capacitor values. For instance, a simple inspection of the evolved circuit shows that reducing the value of C1 is a way to enhance the transfer function, by shifting the lower half of the circuit passing band from 20kHz to 50 kHz.

Finally, we performed a comparison between the performance of our GA and the hillclimbing search technique.[17] A total of 20 GA executions were performed, each one processing 40 individuals along 100 generations. The same number of individuals has been processed in the hillclimbing method, i.e., 20 executions processing one individual along 4000 generations. We note that the GA was optimized in terms of mutation rate (around 1 mutation per genotype) and of selection pressure (exponential selection with parameter c equal to 0.9). The graph of Figure 5.24 shows the average fitness obtained in both experiments. Although the GA outperformed hillclimbing, the latter performed surprisingly well for this problem.

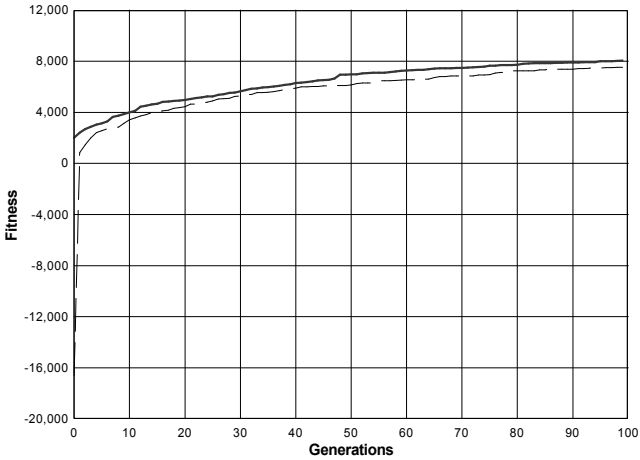


Figure 5.24 Average fitness along the generations for two experiments: GA (full line) and hillclimbing (traces). In the case of hillclimbing, the fitness values were taken within an interval of 40 generations, in order to match the number of 100 fitness points.

5.1.2.4.2 Multi-Objective Experiments: First Case

Although hillclimbing and GAs produced comparable performances in the single-objective experiments, the genetic algorithm is better tailored for the multiple-objective application. This is due to the fact that, as it is described in the third chapter, the EMM multi-objective fitness evaluation method requires the computation of an average over a set of individuals. Since hillclimbing focuses on only one individual at a time, it can not be applied in the context of this technique.

In this first case of multi-objective study, four objectives have been considered: the frequency response fitness, computed as shown in equation 5.1; the minimization of the power dissipation; the maximization of the Maximal Symmetric Excursion (MSE); and the minimization of the integrated output noise. The power dissipation and the integrated output noise are directly measured by the simulator; the MSE is maximized by keeping the DC value of the output voltage, $V_{dc}(out)$, at half of the power supply value (1.5V); this is therefore accomplished by *minimizing* the quantity $|V_{dc}(out) - 1.5|$.

These four objectives are aggregated in the way shown in equation 3.6, and the weights are updated through the expression presented in equation 3.8 (Chapter 3). The design plan (user's specifications) used in this experiment was set to: fitness of the frequency response (equation 5.1) equal to 10000; power consumption equal to 0.1 mW; value of $|V_{dc}(out) - 1.5|$ equal to 0.5V, corresponding to a 1.0V excursion at the output; and integrated output noise equal to –120dB. We ran 10 GA's executions, each one including 40 individuals and 100 generations. The overall experiment lasted around 4 hours in one 300 MHz Ultra Sparc processor. The SMASH[18] simulator was used in this set of experiments. The graphs of Figure 6 display the average values taken by the four objectives during this experiment. It can be seen that the

GA tries to optimize the frequency response, MSE, and dissipation, keeping control of the output noise simultaneously.

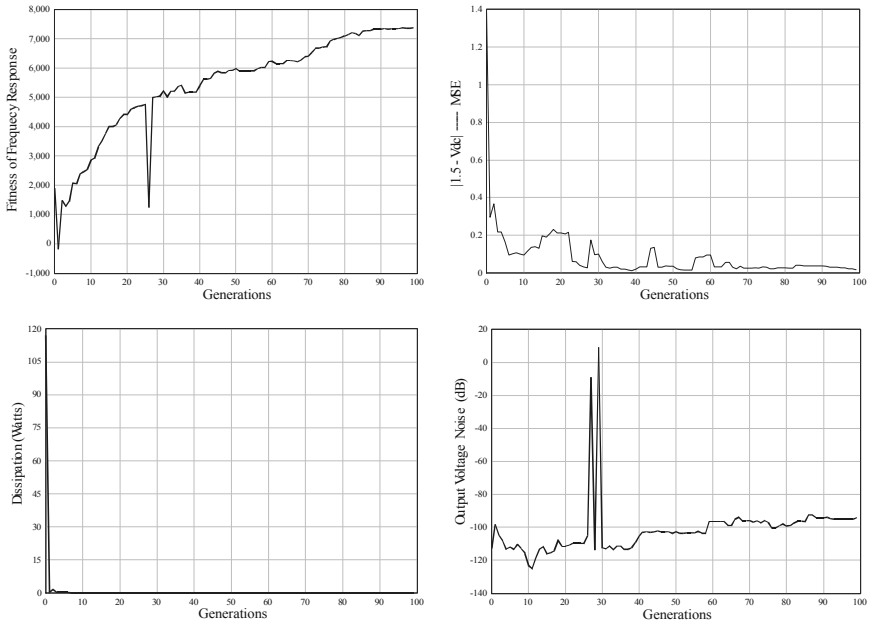


Figure 5.25 Average values of the objectives along the evolutionary process: frequency response(A); MSE (B); Dissipation (C); Noise (D).

Figure 5.26 shows the schematic of the best circuit achieved in this set of experiments.

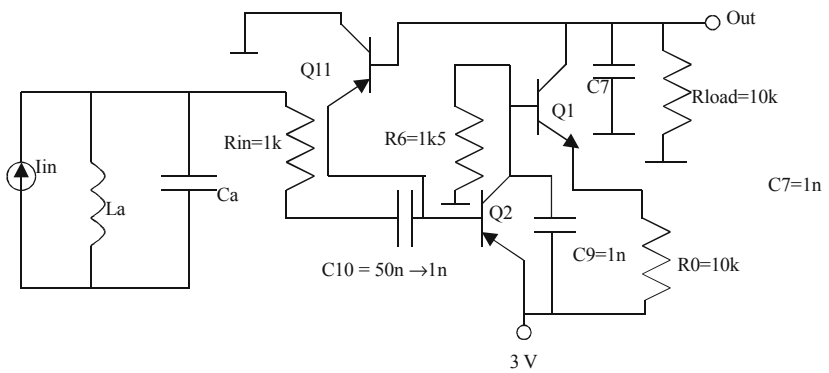


Figure 5.26 Best filter achieved in this set of experiments.

In the above circuit, transistors Q11 and Q2 work in the linear region and transistor Q1 works in the reverse region. Q2 is performing the signal amplification and

delivering it to the pair Q1 and Q11, which is setting the DC output to the desired value. As shown in the schematic, this design can be improved by a simple inspection: the value of C_{10} may be decreased from 50nF to 1nF; and the input impedance of the amplifier may be increased by inserting R_{in}. These changes improve the passing band boundaries within the AM frequency region by shifting it to the right. The graphs of Figure 5.27 show the frequency response of the circuit, with and without these changes.

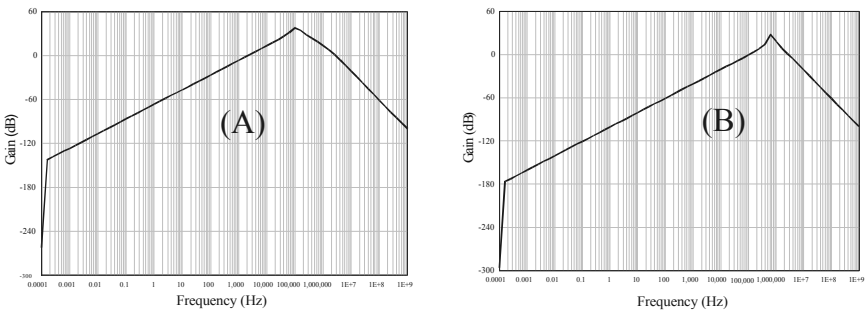


Figure 5.27 Gain of the circuit shown in Figure 5.26: without changes (A); improved design (B).

We can draw a comparison between the performance of the circuit obtained in the first set of experiments and the improved version of the circuit shown in Figure 5.26. Both of them display a gain close to 30dB at the frequency of 500kHz. The design of Figure 5.26 presents an integrated noise at the output of –94 dB, against –83dB observed in the circuit of Figure 5.22. A reference value for noise in bipolar amplifiers can be taken from Johns, D.A., and Martin, K.[1] (1997) where the output noise for a common emitter amplifier, with DC operating point optimized for noise attenuation, assumed a value around –80dB at 300K (the same temperature used in our experiment). The power consumption of the above circuit is 5.0 mW, against 3.9mW of the circuit depicted in Figure 5.22. This difference is due to the fact that the amplifying transistor of the above circuit, Q2, is draining more current from the power supply than the single transistor of the first circuit. Finally, the MSE of the circuit in Figure 5.26 is around 1.5V (DC output value is 1.5V), against 0.9V for the circuit shown in Figure 5.22. Therefore, we can conclude that the second circuit is better in terms of output excursion and noise.

An important issue concerning circuit evolution through simulation is their implementability. It has been verified, in a previous work,[19] that simulators might bias transistors in overvoltage and overcurrent conditions, which could not be reproduced in practice. The minimization of the circuit dissipation along the evolutionary process, accomplished in this experiment, was the means whereby these conditions could be avoided. This issue will be further discussed when we talk about evolutionary design of operational amplifiers.

5.1.2.4.3 Multi-Objective Experiments: Second Case

There is a problem associated with the circuits evolved in the previous sections. When we attempt to optimize the frequency response in the way shown in equation 5.1, we focus only on the circuit gain V_{out}/V_{in}. Nevertheless, we may well obtain a circuit with a high gain, but with very low values for V_{out}. We must remember that the signal is supplied to the amplifier from a trap with resonant frequency at 1MHz. If by accumulating the number of frequency points, inside the passing band, in which the input signal level surpasses –50dB. The higher the signal at the amplifier input, the higher the input impedance of the amplifier (at the AM band) will be. We sample around 10 frequency points in the AM band; hence, the fitness associated with the fifth objective can vary from 0 to about 10 points.

We ran ten executions of the Genetic Algorithm, sampling 40 individuals along 400 generations. The parameter α was set to the value of 0.8. Single-point crossover applied at a rate of 80% and a mutation rate of 2.5% per genotype position have been employed. The design plan was set to: fitness of the frequency response of 9,000; symmetric excursion of 1.0 V; dissipation of 10mW; output noise at –110dB; at least seven points of the AM band where the amplifier input voltage surpasses the value of –50 dB. We do not expect to fully satisfy the requirements, but we hope to induce the GA to obtain solutions close to these specifications. The five graphs of Figure 5.28 show the average values of the objectives along the evolutionary process.

Figure 5.29 presents the graph of the energy associated to the optimization process, averaged over 10 executions. The success of the algorithm in satisfying all the objectives is reflected by the decreasing value of the energy.

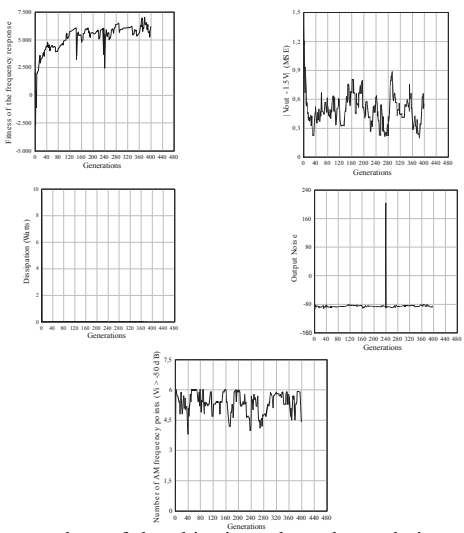


Figure 5.28 Average values of the objectives along the evolutionary process: frequency response(A); MSE (B); Dissipation (C); Noise (D); Number of points at the AM band where the input voltage surpasses the level of –50dB (E).

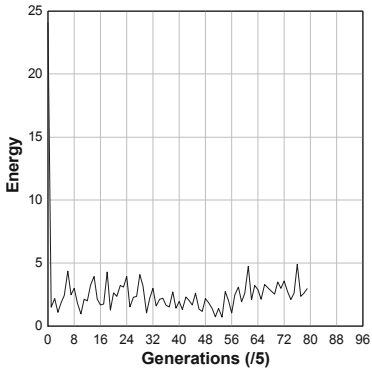


Figure 5.29 Average value of the system energy along the evolutionary process in the third case study of active filter synthesis. Note that the X axis include 80 points, since the energy is updated each 5 generations.

Two circuits achieved in this experiment are depicted by Figure 5.30 (circuit A) and Figure 5.32 (circuit B). Their frequency responses are respectively shown in Figure 5.31 and Figure 5.33.

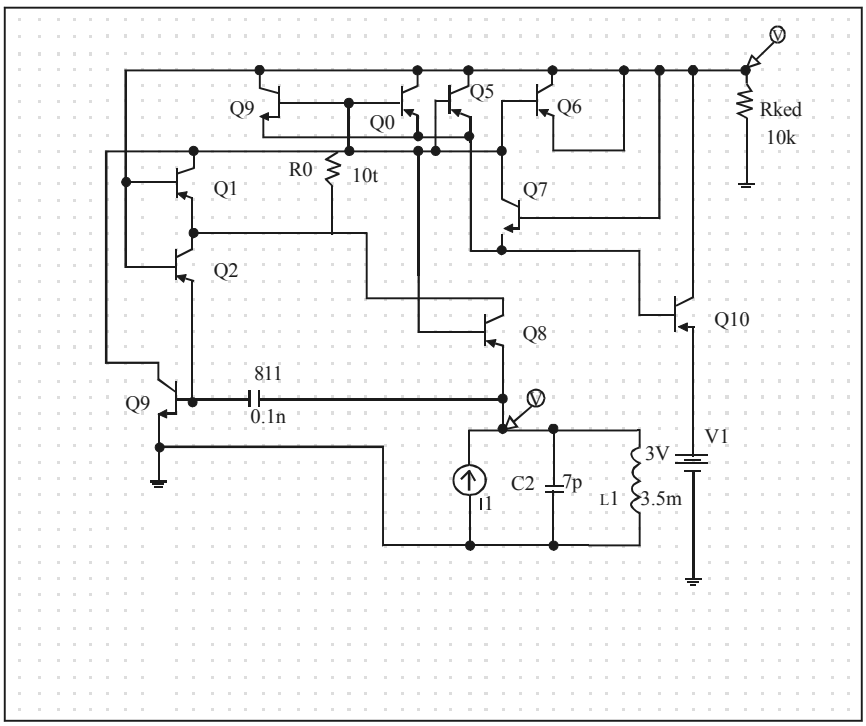


Figure 5.30 One circuit (A) achieved in this set of experiments taking into account multiple objectives.

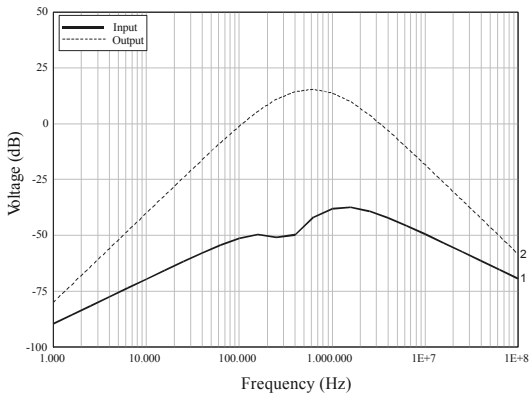


Figure 5.31 Frequency response of circuit A (Figure 5.30). Input and output voltages in decibels.

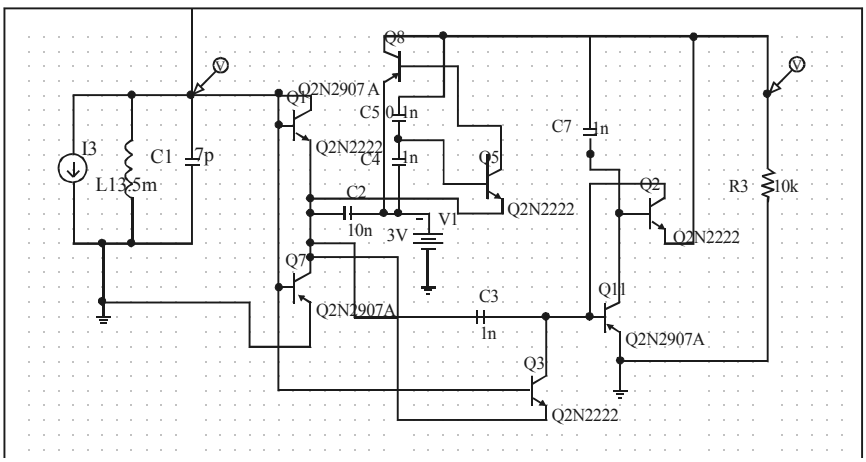


Figure 5.32 One circuit (B) achieved in this set of experiments taking into account multiple objectives.

Table 5.4 compares the performance of the evolved circuits, (A) and (B), and a human designed receiver, shown in Figure 5.33. From this table, it can be seen that circuit A achieves the highest gain among the circuits. Referring to the noise at the their inputs, the evolved circuits display similar performances, as shown in this table. We also try to make a partial comparison of the power dissipated by the circuits. Circuits (A) and (B) display very similar dissipation statistics. We can not perform a fair comparison with the human designed circuit though, because it uses an 8-Volt DC power supply. That is also why the human-designed circuit achieved the highest output swing.

Table 5.4 – Comparison among the performance of the evolved circuits, (A) and (B), shown previously, and a human made design.

	Evolved Circuit (A)	**Evolved Circuit (B)**	**Human Designed**
Maximum Loop Gain	63 dB	32 dB	38dB
Minimum Loop Gain	48 dB	16 dB	14dB
Dissipation	0.18mA x 3V = 0.54mW	0.17mA x 3V = 0.51mW	7.5mA x 8V = 60mW
Noise (input) at 10^5Hz	$1.22 \cdot 10^{-11}$ A/Hz $^{1/2}$	$6 \cdot 10^{-13}$ A/Hz $^{1/2}$	10^{-12} A/Hz $^{1/2}$
Noise (input) at 10^6Hz	$7.62 \cdot 10^{-13}$ A/Hz $^{1/2}$	10^{-12} A/Hz $^{1/2}$	$5 \cdot 10^{-13}$ A/Hz $^{1/2}$
Maximal Output Swing	2.4 V p-p	2.8Vp-p	4V p-p

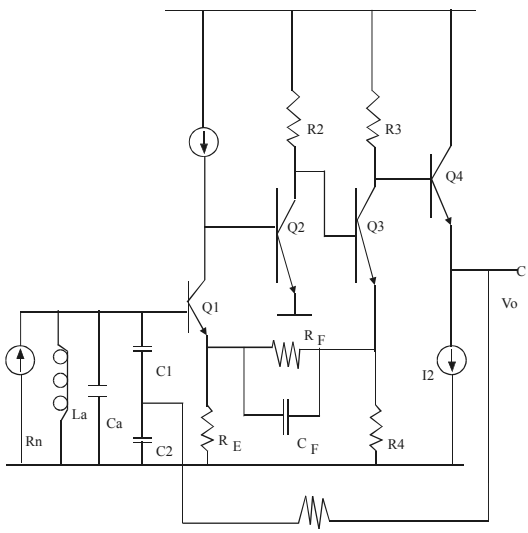


Figure 5.33 Human designed AM receiver.

5.1.3 Synthesis of Operational Amplifiers Based on Bipolar Technology

We analyze in this section the synthesis of operational amplifiers whose building blocks are discrete components: bipolar transistors and resistors. We start with a description of the problem, and then present the main features of the GA utilized, representation, and evaluation. Afterwards, the case studies are presented.

5.1.3.1 Problem Description

Operational amplifiers or OpAmps are the most versatile and widely used building blocks in analog systems. They are directly coupled differential amplifiers, presenting

a high voltage gain, high input impedance, and low output impedance. In order to control their gain and bandwidth, they are usually configured with an external feedback circuit. Nowadays, operational amplifiers are realized in integrated circuits through bipolar, CMOS, and BiCMOS technology. We have already examined the optimization of CMOS and BiCMOS OpAmps in the last chapter. We will now study the synthesis of bipolar OpAmps.

We have already seen, in the last chapter, that OpAmps are usually designed in a very structured way: an input stage consisting of a differential amplifier; an intermediate stage that provides gain to the amplifier; and an optional output stage, usually consisting of a Cascade stage.

Naturally, we can not expect the EA to arrive at circuits structured similarly to human designed OpAmps, even though these topologies are present in the search space. Due to the large search space the EA is going to sample, we expect to arrive at new topologies that human designers would not think of using. Of course, these new solutions should not only represent curiosities, but valuable options to replace conventionally designed circuits. Naturally, this same idea applies to the cases studied previously in this chapter.

5.1.3.2 Representation

We use the same direct representation employed in the case of passive and active filters synthesis. In this particular case, each gene may code for three types of components: npn bipolar transistors; pnp bipolar transistors; and resistors. Capacitors are not used in this case, because we are not particularly interested in the frequency response, as explained in the evaluation section.

5.1.3.3 Evaluation

In chapter 4 we showed that the task of designing OpAmps could be translated to a multi-objective search problem, involving the satisfaction of the following specifications: gain, bandwidth, phase margin, dissipation, and many others. An alternative way to assess OpAmps, suggested by Bennett[20] et al. (1997), is through their DC transfer function. This kind of assessment will not consider all the aspects just mentioned, but only gain, linearity, and DC biasing. However, it is worthwhile to investigate this approach for circuit evaluation, due to its simplicity.

The circuits are then evaluated using the DC transfer mode of the SMASH simulator. This contrasts with the previous case, in which the AC analysis of the simulator was employed.

Figure 5.34 depicts the DC transfer function of an operational amplifier. This graph plots the output voltage as a function of the differential input. We can observe that this curve is divided into three regions: negative saturation region; linear region; and positive saturation region. The linear region lies between the two saturation regions, occupying a narrow range of the differential input voltage, usually around 10mV. There are two saturation voltages, v_s^+ and v_s^-, of positive and

negative polarities respectively. It is usually desirable to have this saturation voltages symmetrical, and as close as possible to the power supplies' values, *Vdd* and *Vss*.[16] The DC transfer function has then served as a basis to the fitness computation.

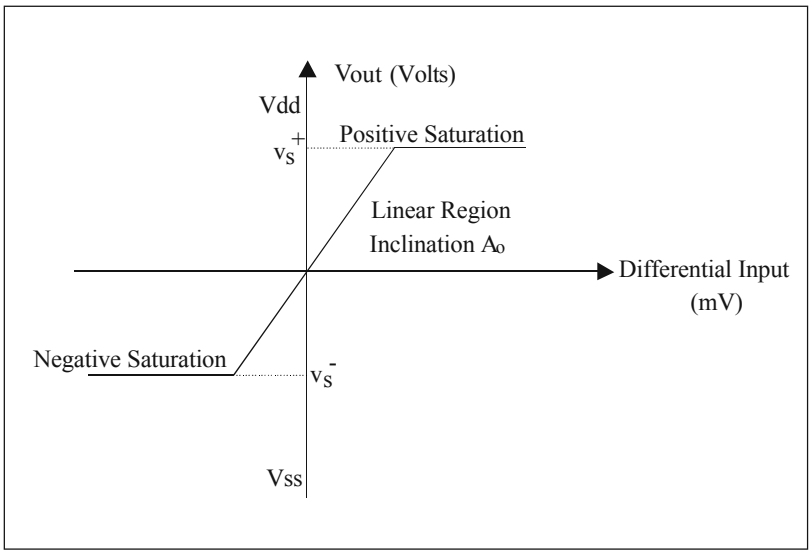


Figure 5.34 Typical DC transfer function of an OpAmp.

As a consequence of the fact that this transfer function is already based on the differential input, the task of common signal rejection, performed by the differential stage, will not be considered by the fitness evaluation function. Two fitness equations were tested in the case studies and their expressions are furnished in the next section.

5.1.3.4 Case Studies

In this section we will describe three sets of experiments that differ by the use of distinct fitness evaluation functions and also by the use of fixed and variable length chromosomes.

The first set of experiments employed a fixed length representation. There are 15 genes per chromosome, and, as defined earlier, each gene may encode for a npn transistor, pnp transistor, or a resistor. In terms of components arrangement, the GA is allowed to use eight interconnecting points, five of them being external ones. The external points are associated to two DC power supplies of 12V and –12V, to the ground, to the differential AC input and to the probed output. The following fitness equation has been used in this first case:

$$Fitness = \left(\left| v_s^+ - v_s^- \right| \right) - A \cdot v_s^+ \cdot v_s^-$$

The first term in the above equation evaluates the difference between the positive

and negative saturation values, accounting for gain of the amplifier. The second term realizes the negativity of the product of the saturation voltages, accounting for the amplifier symmetry. The constant A is used to balance the two objectives. Through much experimentation, a value of A=20 has been found to work better. We employed a DC transfer simulation where the differential input signal varied from –10mV to 10mV[20]. We assumed that the saturation voltages were associated to these two limits.

Another important aspect concerning this experiment is the implementability of the evolved circuits. We can identify some conditions produced by the simulators that would prevent the circuit from working if implemented with low-power transistors. Particularly, the circuit simulator may use bipolar transistors with base-emitter voltages beyond the specifications of low-power transistors. We need to constrain the base-emitter voltages in the following way:

$$-6\ Volts < V_{BE} < 0.9\ Volts, \text{ in the case of npn transistors; and}$$
$$6\ Volts > V_{BE} > -0.9 Volts, \text{ in the case of pnp transistors.}$$

The specifications of the commercial devices BC107 npn and BC177 pnp were used to establish these conditions. Whenever these conditions are violated, the simulator will provide an abnormal high value for the transistor collector current as well.

There are two approaches to handle this constraint: the first one, used in this case study, is to strongly penalize circuits presenting at least one transistor with base-emitter voltage outside the specification; the second one, used in the evolution of the AM active filter, consists in minimizing the power consumption of the circuit. Whenever we minimize the circuit power consumption, we avoid the situations in which the bipolar transistor display an abnormal collector current.

Ten executions of a GA were executed, which processed 40 individuals along 200 generations. Figure 5.35 displays the best circuit evolved in this experiment.

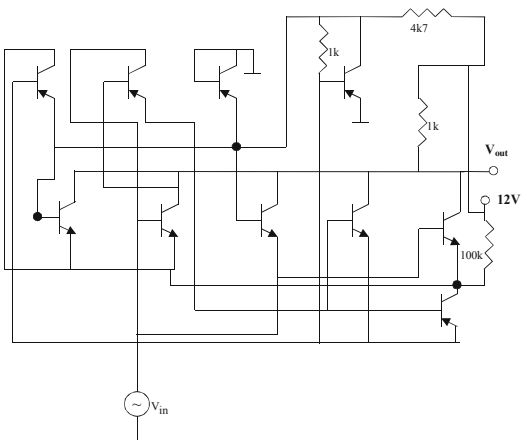


Figure 5.35 Schematics of the best circuit achieved by the first set of experiments.

The amplifier shown in the above figure displays a DC gain of 39 dB and a dissipation of 59 mW. The resistor values evolved by the algorithm have been slightly changed to other values currently available. Figure 5.36 compares the DC transfer obtained in the simulation with the measured response of the same circuit implemented through discrete devices.

Figure 5.37 shows the circuit frequency response to a load of 0.1 μF. We can observe that the amplifier behaves as a low-pass filter that presents a gain-bandwidth product of 100 kHz.

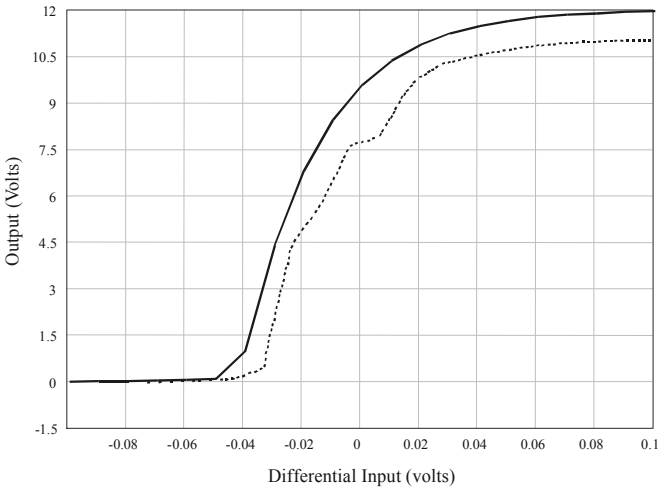


Figure 5.36 DC transfer characteristic of the amplifier evolved in the first set of experiments: full line accounts for the result obtained in simulation, and traced line represents the response measured in the implemented circuit.

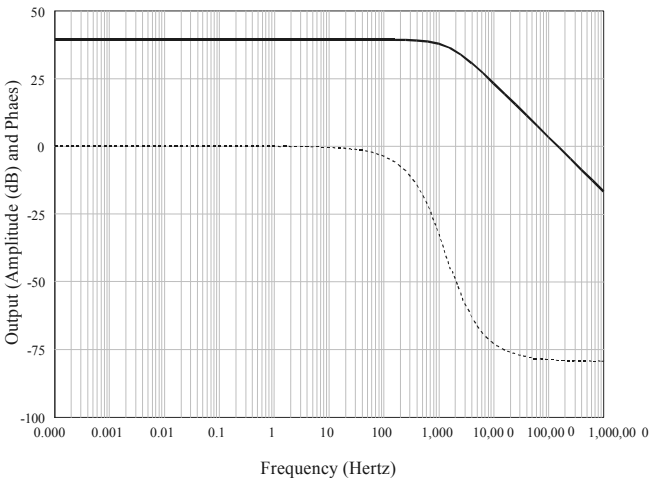


Figure 5.37 Frequency response of the amplifier evolved in the first phase of the experiments. A capacitive load of 0.1 μF was connected to the output. Amplitude, measured in decibels, in full line; phase in points.

The results achieved are interesting, mainly due to the fact that the evolved circuits could be implemented through discrete components. Nonetheless, all of the evolved circuits lacked a symmetrical DC response. For instance, in the circuit of Figure 5.35, the evolved circuit did not even use the power supply of –12 Volts, working only in the positive region. If we forced all the circuits to use both power supplies, this situation could change. But we preferred to follow another strategy, characterized by two new procedures: conceiving another fitness evaluation function; and setting the differential signal reference terminal to –12V.

The new fitness evaluation function is also based on a DC transfer, in which the input signal is swept through the power supply boundaries, –12V and 12V, within a 100mV step. This function is described by the following equation:

$$Fitness = \left[Máx_{i=1}^{n-1} \left(\left| V(i+1) - V(i) \right| \right) \right]$$

In the above equation, *V(i)* stands for the circuit output voltage as a function of the DC analysis index *i*. The fitness is then given by the maximum absolute difference between consecutive output voltage samples. The main distinction of this procedure when compared to the former fitness evaluation function relies on the fact that DC analysis includes now the overall voltage spectrum of the circuits, determined by the power supplies.

The second change is not really obvious at a first glance. In the first set of experiments, it has been verified that the GA performance is very sensible to the differential signal DC level, which determines the DC operating point. Through experimentation, it has been verified that the performance greatly improved when we fixed the differential signal DC level to –12 Volts, instead of grounding it.

In this second set of tests we have employed, again, fixed length representation. Each chromosome is composed of eight genes, and there are a total of 10 possible connecting points. This experiment sampled around 10^5 individuals, taking about 20 hours in a Sun workstation *Ultra Enterprise 2*.

The best amplifier evolved in this experiment is depicted in Figure 5.38. The topology displayed in Figure 5.39 is not the kind that would be expected from a human designer: the differential input is applied to the collector, and not to the basis of transistor Q1; and, though not explicit in the figure, we verified that transistor Q3 is redundant to the amplifier. Through an operating point analysis, we observed that transistors Q2 and Q4 work as current amplifiers, whereas Q1 works in the reverse region. Q1 and Q2 set the DC operating point of transistor Q4 to an adequate value, and this last transistor provides the amplification effect. Therefore, it is a single stage amplifier. A diode has been inserted to the final circuit, with the purpose of removing a 0.7V bias voltage, increasing, thus, the maximal symmetrical excursion. Besides this diode insertion, some human knowledge has also been utilized to adjust the resistors' values, aiming to improve the amplifier linearity. Such an interaction with an expert is not necessarily undesirable, since the objective is to enhance an evolved circuit, and not to bias the evolutionary algorithm to achieve human-like designs.

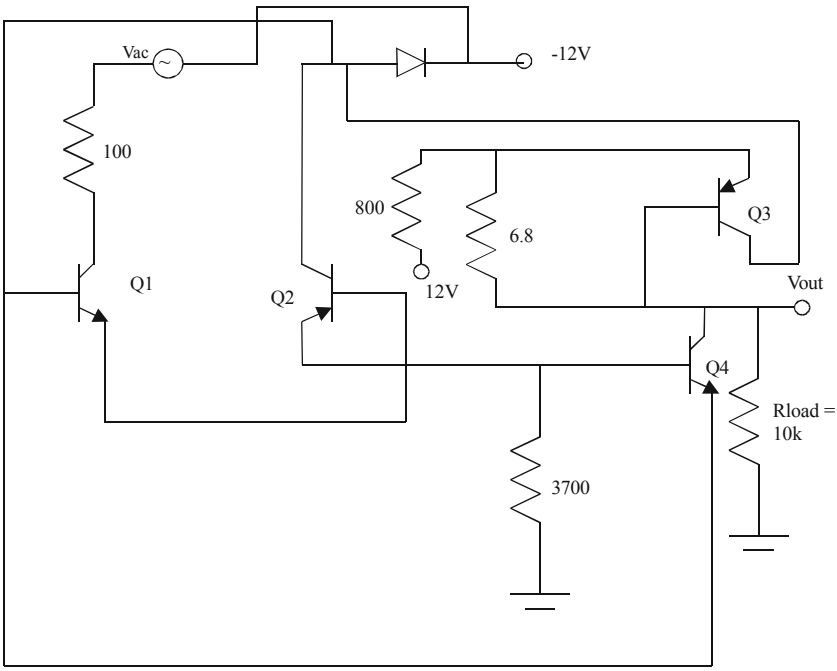


Figure 5.38 Best amplifier evolved in this set of experiments.

The DC transfer of the circuit in Figure 5.38, depicted in Figure 5.39, is more satisfactory than the one achieved in the first experiment, since the saturation values are closer to the DC power supply values. We can also observe, from the frequency response shown in Figure 5.40, that this amplifier behaves as a low-pass filter, with a gain of 53 dB. The dissipation stays around 0.1W. This circuit has been realized by the use of discrete components, displaying a behavior close to the one predicted in simulation.

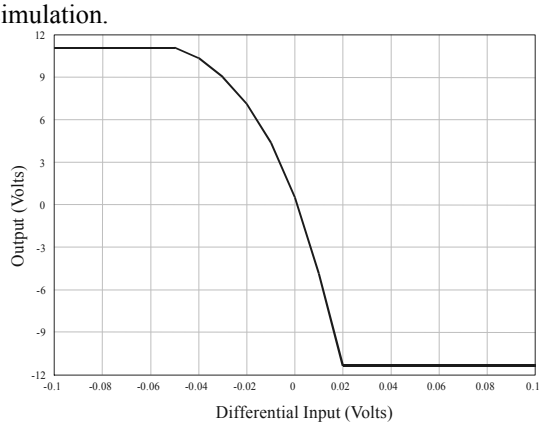


Figure 5.39 DC transfer of the amplifier shown in Figure 5.42.

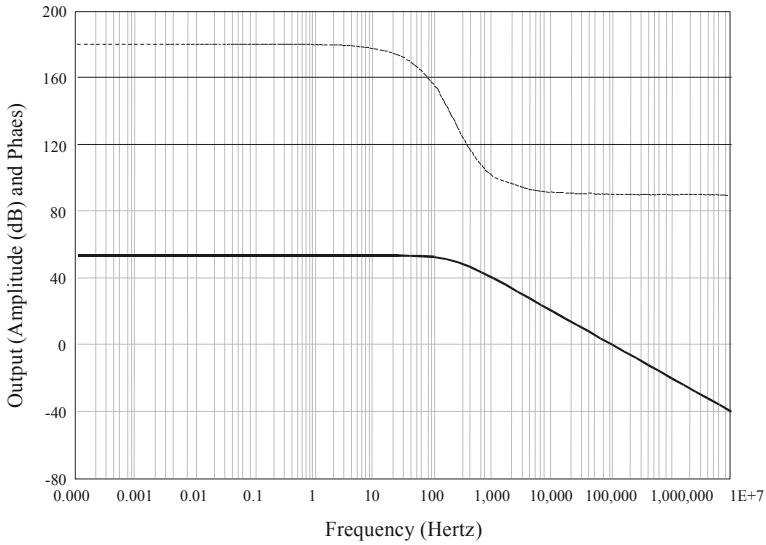


Figure 5.40 Frequency response of the amplifier shown in Figure 5.38. A capacitive load of 0.1 µF was connected to the output. Amplitude, measured in decibels, in full line; phase in points.

5.1.4 Synthesis of a Digital to Analog Converter (DAC)

The evolution of digital to analog converters is another relevant case in . Data converters perform the interface between the digital world and the real analog world. DACs have a widespread use in the area of digital audio, where the digital information stored in the CD is converted into music via high-precision converters. The synthesis of DACs includes several requirements, such as speed, linearity of the transfer curve, accuracy, and resolution. Most of the DACs presented in the literature are decoder based, in which a string of resistors is used to generate 2^N reference signals, where N is the number of digital inputs. The digital input word is then used to select the appropriate reference signal. The evolutionary approach for circuit synthesis, on the other hand, usually explores alternate ways to synthesize circuits that are rather different from human designed ones.[21]

GP has been applied to the evolution of DACs.[22] The representation was based on the circuit programming tree, described previously in this chapter when GP was applied to evolve passive circuits. In this case, the components available are bipolar transistors, capacitors, and resistors. This particular application consisted of the evolution of a 3 bit DAC.

The fitness was computed through the execution of transient analysis in the SPICE simulator. There is a total of eight fitness cases, consisting of the eight possible configurations of the three input bits. Each fitness case was presented twice during the evaluation function: the inputs were presented first in an up-counting sequence and after in a down counting sequence. This procedure is desirable to avoid effects of the initial state, i.e., an evolved circuit may work correctly

when the inputs are applied in a particular sequence, but incorrectly when another sequence is presented! This is certainly a drawback of evolved designs contrasting to conventional designs. Of course, DACs must work correctly when the inputs are applied in any different sequence. Naturally, it would be unfeasible to evaluate all the possible configurations of the input bitstring in the transient analysis; hence, the procedure is to select only some cases. If the circuit works correctly for more than one case, it will possibly work correctly in all different sequences. Another way to overcome this problem is to perform an operating point analysis for each different input configuration, so that we will be sure that the observed response is not only a transient behavior.

Figure 5.41 displays the best DAC achieved by Bennett[26] et al. (1999), as well as the circuit response. This circuit was achieved after 139 generations using a population size of 330,000.

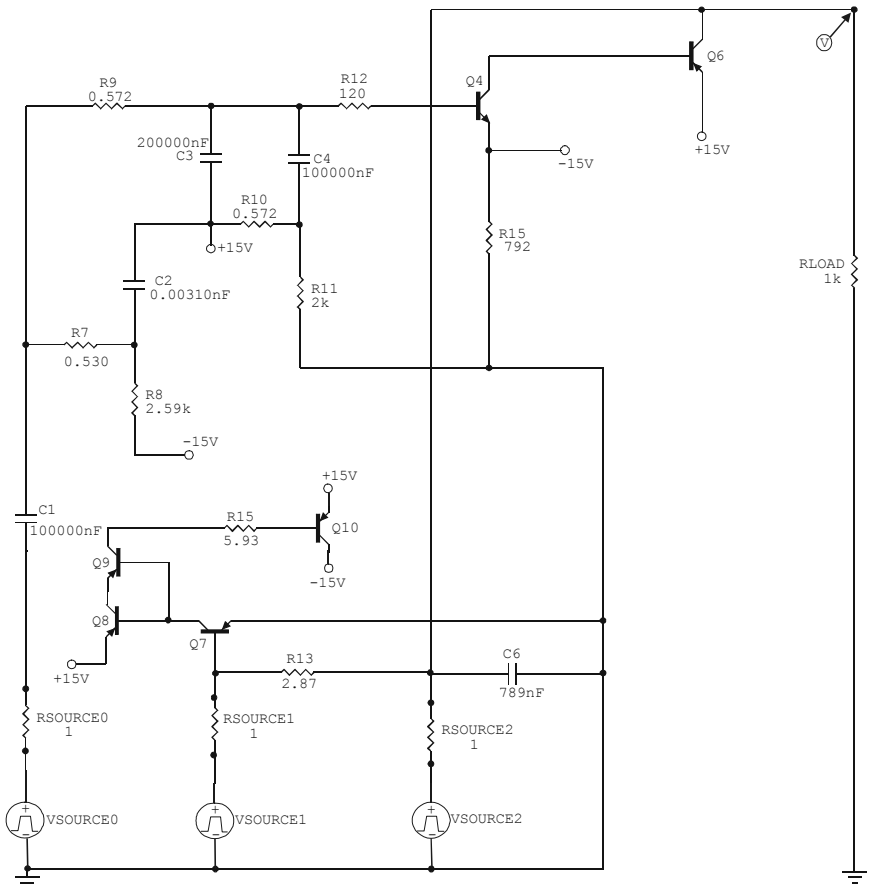


Figure 5.41a DAC evolved through Genetic Programming and circuit response. (Extracted from Bennett[22] et al. Figs. 9 and 10.)

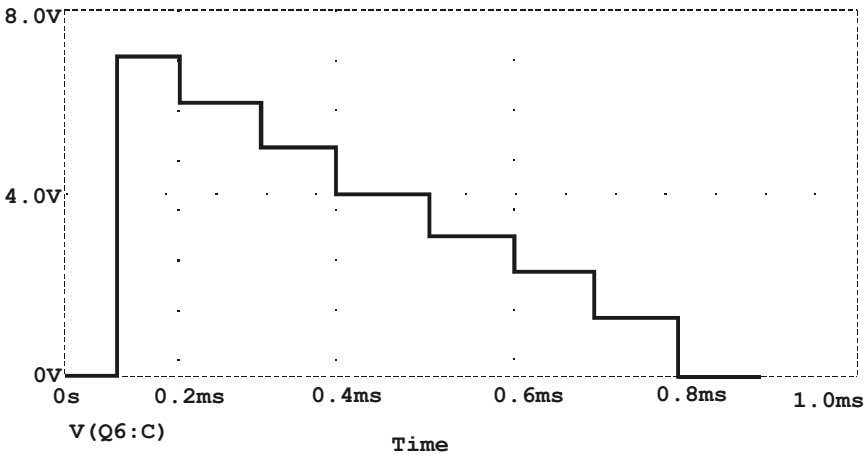


Figure 5.41b DAC evolved through Genetic Programming and circuit response. (Extracted from Bennett[22] et al. Figs. 9 and 10.)

REFERENCES

[1] Johns, D.A. and Martin, K., *Analog Integrated Circuit Design*, John Wiley & Sons, New York, 1997.

[2] Laker, K.R. and Sansen, W., *Design of Analog Integrated Circuits and Systems*, Mc Graw-Hill, New York, 1994.

[3] Lohn, J.D. and Colombano, S.P., Automated analog circuit synthesis using a linear representation, in *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, Lausanne, September, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 125.

[4] Sussman, G.J. and Stallman, R.M., Heuristic techniques in computer-aided circuit analysis, *IEEE Transactions on Circuits and Systems*, vol. 22, 1975.

[5] Harjani, R., Rutembar, R.A., and Carey, L.R., A prototype framework for knowledge-based analog circuit synthesis, in *Proceedings of 24th Design Automation Conference*, 1987.

[6] Ochotta, E. S., Rutenbar, R. A., and Carley, L. R., Synthesis of high-performance analog circuits in ASTRX/OBLX, *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 273-294, 1996.

[7] Grimbleby, J.B., Automatic analogue network synthesis using genetic algorithms, in *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems (GALESIAS - 95)*, England, 1995, 53.

[8] Horrocks, D.H. and Spittle, M.C., Component value selection for active filters using genetic algorithms, *First On-line Workshop on Soft Computing (WSC1)*, Special Session on , August 1996, 19.

[9] Ellis, M., *Electronic Filter Analysis and Synthesis*, Artech House Inc., 1994.

[10] Koza, J.R., Bennett III, F.H., Andre, D., and Keane, M.A, Toward evolution of electronic animals using genetic programming, in *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, MIT Press, Cambridge, 1996.

[11] Bishop, O., *Practical Electronic Filters*, Babani Electronics Books, 1991.

[12] Koza, J.R., Bennett III, F.H., Forrest, H.K., Martin, A., and Andre, D., Evolution of a time-optimal fly-to controller circuit using genetic programming, Koza, J.R. et al., Eds., *Genetic Programming 1997: Proceedings of the Second Annual Conference*, July 13-16, 1997, Stanford University, San Francisco, Morgan Kaufmann, San Manteo, CA, 207.

[13] Koza, J.R., Bennett III, F.H., Andre, D., and Keane, M.A., Four problems for which a computer problem evolved by genetic programming is competitive with human performance, in *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press, 1.

[14] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers, in *Proceedings of the XIII International Conference in Microelectronics and Packaging*, Vol. 1, Chueiri, I.J. and Reis, C.A.F., Eds., Curitiba, Brazil, 264.

[15] Sansen, W., Low-noise wide-band amplifiers in bipolar and CMOS technologies, in notes on the course Low-noise analog CMOS & BiCMOS design, promoted by the Imperial College, February 25-27, London, England.

[16] Laker, K.R. and Sansen, W., Eds., *Design of Analog Integrated Circuits and Systems*, Mc Graw-Hill, Singapore, 1994.

[17] Blickle, T., Theory of evolutionary algorithms and application to system synthesis, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1996.

[18] *SMASH User and Reference Manual*, Dolphin Integration, France, 1993.

[19] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., Synthesis of CMOS operational amplifiers through genetic algorithms, in *Proceedings of the Brazilian Symposium on Integrated Circuits (SBCCI98)*, IEEE, Rio de Janeiro, Brazil, September 1998, 125.

[20] Bennett III, F.H., Koza, J.R., Andre, D., and Keane, M.A., Evolution of a 60 decibel op amp using genetic programming, in *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, vol. 1259, LNCS, 1997, Tsukuba, Japan, Springer-Verlag, October 1996, 455.

[21] Zebulum, R.S., Astoica, A., and Keymeulen, D., Experiments on the evolution of digital to analog converters, in 2001 IEEE Aerospace conference (CD-ROM), March 2001, Montana.

[22] Bennett III, F.H., Koza, J.R., Keane, M.A., Yu, J., Mydlowec, W., and Stiffelman, O., Evolution by means of genetic programming of analog circuits that perform digital functions, Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., Eds., in *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference*, July 13-17, Orlando, FL, Morgan Kaufmann, San Francisco, 1999, 1477.

CHAPTER 6

Evolutionary Computation: A Tool for Digital Circuit Synthesis

This chapter complements the case studies on the evolution of analog circuits presented in Chapter 5, reporting experiments on the evolution of digital circuits. We start by describing three possible approaches for digital circuit representation: functional level, gate level, and transistor level representations. We then continue with the description of the fitness evaluation functions. Afterwards, the following case studies are presented: the synthesis of combinational circuits; the synthesis of sequential circuits; the synthesis of transistor logic; and the synthesis of digital filters.

The idea of using evolutionary algorithms to synthesize circuits' topologies, first proposed by Louis and Rawlins[1] (1991), has been initially tested in the area of digital circuits. There are many reasons for this: digital systems are more widely used than analog ones; their simulation is usually more reliable than the simulation of analog circuits; and, most importantly, the appearance of high performance reconfigurable digital circuits that can be used as a target implementation device for the evolved circuits.

Let us concentrate on the last reason, the appearance of high performance reconfigurable digital chips. This topic will be further developed in the next chapter, so we just intend to give a superficial description here. The last generations of reconfigurable digital chips, besides displaying a very short reconfiguration time, are also very flexible, in the sense that many different arrangements of digital gates can be implemented. These different arrangements include, for instance, the combination of AND-OR and AND-XOR logic. Reconfigurable chips allow the implementation of sequential as well as combinational circuits. These chips are very attractive to implement digital circuits synthesized through evolutionary systems, since the latter usually consist of unconventional arrangements of logic gates.

The main problem of applying evolutionary systems to the synthesis of digital circuits is the fact that there are already very efficient automatic CAD tools for digital design, such as ESPRESSO and SIS[15]. Therefore, even though the synthesis of digital circuits is an interesting test-bed to compare the performance of different evolutionary algorithms, they have often presented an inferior performance when compared to conventional tools. This stems from the fact that, when using logic paradigms developed by humans, such as the sum of products, there are usually heuristics that can find optimal circuits. These heuristics are most often incorporated into the previously mentioned CAD software. However, there are three points that stimulate us to investigate the evolutionary design of digital circuits:

- We can not say that human devised logic paradigms, such as the sum of products, always provide the optimal solution.

- Given the above affirmative, it is very likely that a tool which is able to explore the whole domain of circuit topologies, including the unconventional ones, will outperform human created heuristics.

- Whenever dealing with technological constraints, such as some kind of limitation in the chip resources, traditional techniques may not be able to find an optimal solution to the problem.

This chapter will develop the first two arguments. The third reason refers to a particular kind of digital design where, according to the nature of the logic gate, there could be restrictions on the maximum number of gates that can be used. For instance, suppose that we have as specification the design of a multiplexer using no more than say five 3-input NAND gates and two inverters, due to availability of chip resources. It is possible that GAs can handle these constraints more efficiently than conventional tools.

We start this chapter by analyzing three possible representations of digital circuits: functional level; gate level; and transistor level representations. These representations differ in the degree of complexity of the circuit basic building blocks. We proceed by describing the circuit evaluation methods employed, and, afterwards, four case studies are described: combinational circuits evolution; sequential circuits evolution; the evolution of circuits based on transistor logic; and the evolution of digital filters.

6.1 REPRESENTATION

6.1.1 Functional Level Representation

The concept of functional level representation of digital circuits has been introduced by Liu et al.[2] (1997). This type of representation corresponds to a high level mapping of digital circuits, and, of course, many different schemes can be thought. This section is organized as follows: we first present a high level representation based on the *sum of products* paradigm, which is also used by the CAD tool Expresso; we then present two other functional level representations, where the building blocks consist of Arithmetic Logic Units (ALU) and multiplexers.

Sum of Products Representation

Considering that any Boolean function can be represented as a sum of product terms, we have conceived the representation of the genotypes as a collection of genes that encodes any possible product of Boolean variables, complete or incomplete. For a *n-input* digital circuit, each gene will have *n* loci, which can take three possible values: **0** if the correspondent input variable is complemented, **1** if the input variable is in the original form (not complemented), and **2** if the input variable is absent. This can be seen in Figure 6.1, which shows the representation

of a hypothetical Boolean function of three variables (a, b, and c), consisting of three product terms, resulting in a chromosome made up of three genes, each one with three loci. As one can see, the number of **2***s* in a Boolean term determines its parsimony. Therefore, we will call the order of a Boolean term as the number of **2***s* in its correspondent gene, so that higher order genes are the most parsimonious ones. The use of variable length representation in this problem is a natural choice, since one does not know, in principle, how many terms the minimal solution is composed of, i.e., the degree of simplification of the Boolean function.[3]

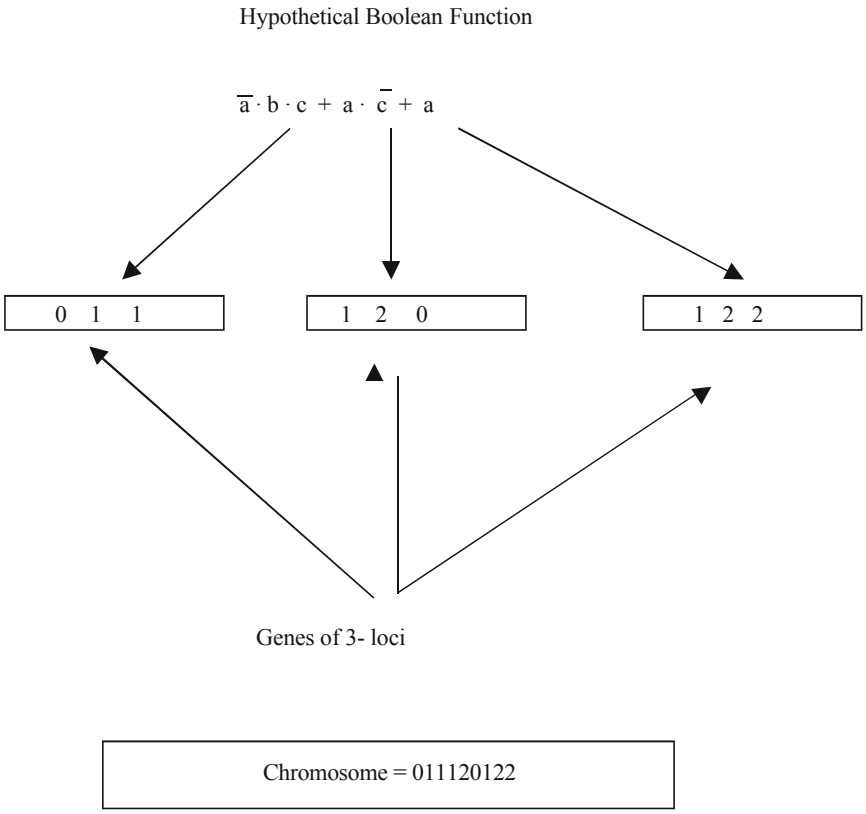


Figure 6.1 Functional level representation based on the sum of products paradigm: hypothetical example of Boolean expression and its chromosome representation.

Since the alphabet cardinality of this representation equals 3, there is a total of 3^n genes in the search space, where *n* is the total number of inputs. The GA must search for the set of genes that complies with a particular circuit specification, given in the form of a truth table[4] that solves the problem in an optimal way. Given a particular truth table, there are usually many different Boolean expressions that satisfy the specification, but only one that solves the problem in an optimal way.

This representation is adequate for the application of the advanced paradigms

defined in Chapter 3. As stated previously, the variable length representation is a natural choice. Additionally, the concept of genetic memory can also be used, since the fitness of each gene, as an individual unit, can be assessed. Finally, this is a typical example of a non-trivial multimodal problem, due to the following reasons:

- Solutions to combinational functions of practical interest usually consist of many genes.
- These genes are usually sampled by different chromosomes along the evolutionary process, i.e., the task of gathering together the desired genes in only one chromosome strongly increases the complexity of the problem.
- Genes that solve a problem may be distant from one another (using the criteria of genetic distance), being then located at distant peaks in the fitness landscape.
- A set of genes that is essential to solve a particular problem usually presents very different individual fitness, being then located at peaks of different heights in the fitness landscape.

Figure 6.2 clarifies the issues mentioned above. The fitness landscape of a 4-input Boolean function, whose optimal solution consists of two product terms, is illustrated. The first product term, consisting of only the input variable a, scores a higher fitness than the second product term, which consists of the product $b \cdot c \cdot d$. As it will be described in the next section, this is caused by the fact that the first product term, being a more parsimonious one, covers more cases of the truth table than the second term. The corresponding genes for these two products, (1 2 2 2) and (2 1 1 1) respectively, are not close to one another, since their symbols are different in all four loci. Furthermore, they are located at peaks of different heights, thus illustrating the problem discussed above. In this particular case, the gene (1 2 2 2) will be more easily found than the gene (2 1 1 1), due to its higher fitness.

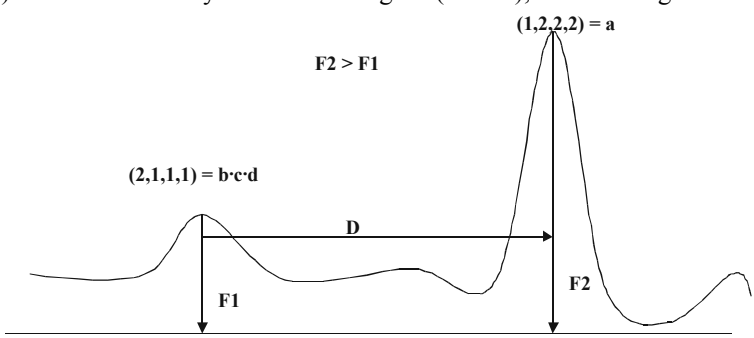


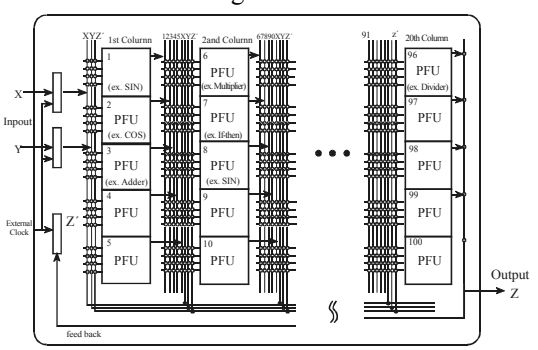
Figure 6.2 Fitness landscape for a particular combinational function. D is the distance between the two solutions. F1 and F2 are the peak heights (fitness) of the two genes (2,1,1,1) and (1,2,2,2) respectively.

ALU Representation

Liu et al.[2] (1997) describes another model for functional level representation of digital circuits based on floating point units:

> *The proposed function-level EHW (Evolvable Hardware) is constructed by FPGAs. The FPGA model contains 100 Programmable Floating Point Units (PFUs) that are arranged in a 20 * 5 grid. Each PFU may perform one of high-level functions, such as addition, subtraction, multiplication, division, sine, cosine, and if-then. The selection of the function to be implemented by a PFU is determined by chromosome genes given to the PFU. Constant generators are also included in each PFU. Neighboring columns of PFUs are interconnected by* crossbar switches. *An output of a PFU will be fed into the input of a PFU located in the next neighboring column.*
>
> ....
>
> *An allele in a chromosome may involve several elements, a template looks like:*
>
> *(N, op, d1, d2)*
>
> *where N is the identity of a PFU (expressed with a number) on which the function op is carried out, di (i = 1,2) can be an input value, a constant or a PFU (identified by its number) from which the input operand is obtained.*

Figure 6.3 provides the schematics of a matrix of PFUs that is programmed by the chromosome previously defined. This reconfigurable system has been tested in simulations, and some interesting tasks have been implemented, such as data compression and ATM cell scheduling.



PFU: Programmable Floating processing Unit

Figure 6.3 Functional level representation based on Programmable Floating Processing Units (PFU). (Extracted from Liu et al.[2] (1997) ICES96, pp. 183.)

Multiplexer Representation

This representation has been devised by Aguirre[5] et al . (1999). This approach is at an intermediate point between the gate level and the function level. The circuit building block is the binary multiplexer with one control line. This device is an universal logic unit, in the sense that any Boolean function of n variables can be implemented by $2^n - 1$ 1-control signal multiplexers. According to Aguirre[5] et al .(1999), this approach is advantageous from the point of view of VLSI design: the manufacturing costs benefit more from the possibility of replicating the same unit than from minimizing the total number of components. This is particularly true in the case of ASICs, but not in the case of reconfigurable chips.

A Genetic Programming representation based on binary trees (or list) is employed. The chromosome consists of a list that encodes a tree-arrangement of a multiplexer.

(A2, (B1, (A0, 0, 1), 0), (B0, (B1, 0, 1), (B1, 1, 0)))

C2	C1	C0	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

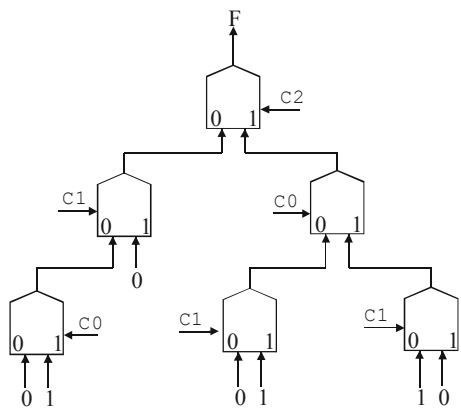


Figure 6.4 Example of Multiplexer Based Representation (Extracted from Aguirre et al. (1999). (NASA workshop 1999, pg. 49, Fig. 4.)

According to the schematic of the figure above, it can be observed that the list encodes the following circuit features:

- Type of multiplexer: class A or B. They differ from which of the two MUX inputs they select when different logic levels are applied to the control line.
- External input that is applied to the control line of each MUX (C0, C1, and C2 in the figure).
- Inputs of the multiplexers, encoded as sub-trees, chosen from the previous tree level. The first level of MUX takes only the logic values “0”and “1”as inputs. This representation has been applied to the evolution of some arbitrary Boolean functions of up to six input variables.[6]

6.1.2 Gate Level Representation

The gate level representation provides a lower level mapping between the chromosome and the digital circuit. The chromosome encodes the arrangement of logic gates, such as AND, OR, NAND, NOR, and XOR, that make up the circuitry. The literature reports many approaches to gate level encoding.[7,8] Figure 6.5 depicts a particular representation proposed by the authors, which is similar, in many respects, to other approaches described in the literature. This figure provides a hypothetical chromosome-circuit mapping.

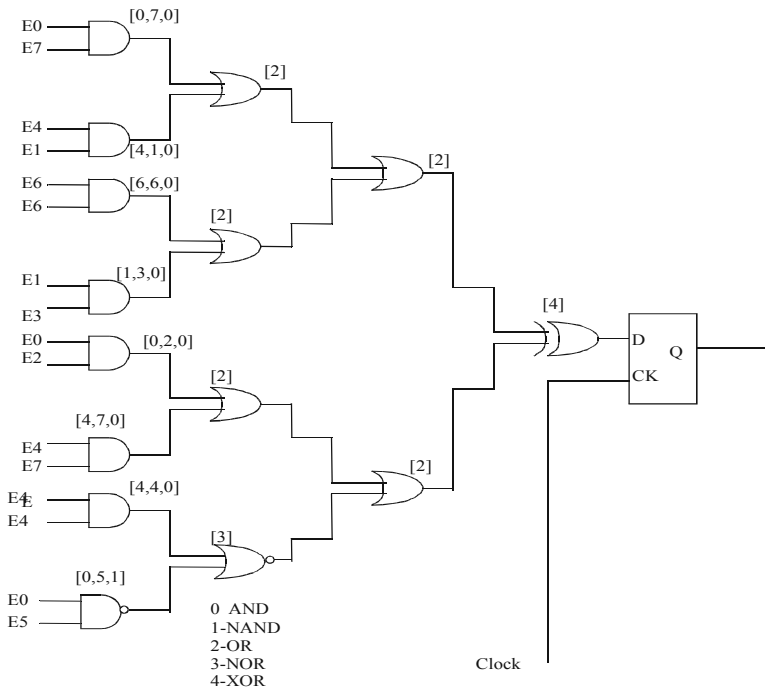


Figure 6.5 Gate Level Representation of Digital Circuits. Genes associated with the gates of the first level are in the format [Input 1; Input 2; Gate Nature]; genes associated with other gates are in the format [Gate Nature].

As can be seen from Figure 6.5, an integer representation is being employed. In this particular example, a predefined structure composed of layers must be evolved. In addition, only 2-input gates are being used. Each gene of the chromosome encodes a particular gate of the circuit: it states the origin of the two inputs and the nature of the gate for the circuit's first layer; and it states only the nature of the gate for the other layers. Five distinct logic gates can be used: AND; NAND; OR; NOR; and XOR. The topology shown in Figure 6.5 encloses one cell of the overall circuit encoded by the chromosome, meaning that each individual is made up of one or more sub-circuits with the structure shown above. The total number of cells is a representation parameter. In the above figure, there are eight hypothetical inputs,

from E0 to E7. These inputs may come from three different sources: external signals applied to the circuit; feedback signals coming from the D flip-flop of the same cell; and feedback signals coming from the D flip-flop of other cells.

This representation provides the following interesting features:

- Both combinational and sequential circuits can be encoded. In the case of pure combinational circuits, the D flip-flop is not considered.
- There is no feedback path in the combinational part of the cell. This would turn the circuit simulation very complex. The feedback connections, originated from the D flip-flops, are controlled by a clock, as in standard sequential circuit design.
- It is a very flexible representation, allowing the implementation of a wide variety of combinational and sequential functions. Opposing the sum of products representation, this scheme also includes the possibility of using XOR logic gates.
- The user may change the depth (number of layers) of the circuit. In the above example, there are four layers. Depending on the particular application, it may be desirable to use only two or three levels, due to speed requirements.
- According to the complexity of the application, the total number of cells that make up the circuit can be changed as well.
- Referring to implementation issues, this particular circuit model can be easily downloaded onto reconfigurable chips. Particularly, as it will be described next chapter, this representation suits very well to the Field Programmable Gate Arrays designed by Xilinx.

Comparing this representation with the functional level representation based on the sum of products, described previously, we can list the following comments:

- The gate level representation has not yet been successfully applied to the evolution of large combinational functions. In contrast, the sum of products representation promises to be suitable when the problem scales up, as it will be shown.
- The gate level representation is more promising in finding circuits that can not be designed through human heuristics. In some cases, this can lead to more efficient circuits.

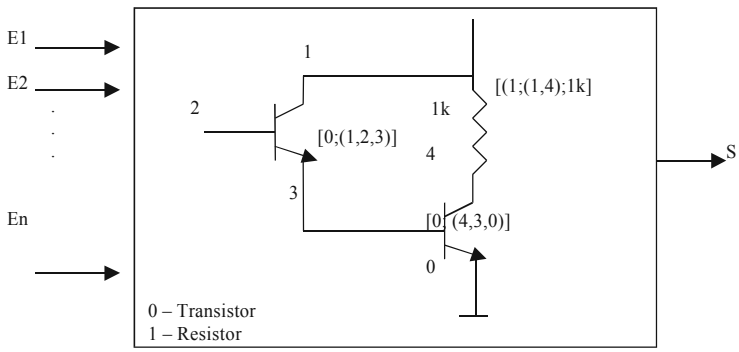
- There is a particular class of Boolean functions for which the decomposition into products connected by XOR logic,[9] instead of OR, results in a more economic implementation. The sum of products representation is not efficient for this class of functions.

- As previously mentioned, the gate level representation is more naturally suited to Field Programmable Gate Arrays (FPGAs) applications, where the digital circuitry mapped by the genotype follows a particular class of FPGA architecture.[10]

6.1.3 Transistor Level Representation

This is the lowest level representation for digital circuits, if compared with the functional and gate levels. It is employed in experiments whose objective is the evolution of a digital behavior through the use of analog components, such as transistors and resistors. Since analog components are the GA building blocks, the same representation described in chapter 5 will be used.

Figure 6.6 depicts an example of the transistor level representation.



Gene = [Type of Component; Connections; Value]

Figure 6.6 Transistor level representation for digital circuits.

As described in the last chapter, the chromosome is made up of genes that encode for each particular circuit component. The gene will state the component's nature, value, and connecting points. Our main interest in this representation is the evolution of digital circuits based on bipolar transistor logic, in a similar way to the well-known *Transistor-Transistor-Logic (TTL)* technology.[4] We will then restrict the possible component nature to bipolar transistors and resistors. In a similar way to the analog circuit representation, the interconnecting points can be classified as internal and external ones. In this particular case, the external points are associated with: the digital inputs (*E1, E2, ..., En* in the circuit of Figure 6.6), the output *S*, or with 5V or 0V voltage levels. The internal points do not have any specific function in the circuit.

6.2 FITNESS EVALUATION FUNCTION

Compared to analog circuits, the simulation, and consequently the evaluation, of digital circuits involves less computation, being faster in terms of speed. In addition, they can be easily accomplished with handwritten simulators.

Different evaluation procedures can be employed, depending on the representation level utilized by the GA. This section describes the evaluation procedures relative to the three representation levels described previously.

Functional Level

The evaluation function, in this representation level, considers the circuit performance through the number of hits in the circuit output, compared to the specified truth table. For instance, in the case of a circuit with *n* inputs, the objective is to evolve a circuit that can attain 2^n hits, corresponding to the size of the truth table. Hence, in a general way:

$$Fitness = Hits, \qquad 0 \le Hits \le 2^n \tag{6.1}$$

However, this evaluation function may be misleading. Whenever evolving digital functions, the objective is to achieve a fully compliant circuit, where the fitness equals the maximum number of hits. For instance, circuits achieving a performance of 99% hits have no interest to the designer. It is very easy for the GA to become trapped in this kind of local optima. In order to avoid these pitfalls, it is necessary to incorporate other terms into the fitness evaluation function. In the case of the *sum of products* representation, the fitness function is changed to:

$$Fitness = Hits - k \cdot Penalties \tag{6.2}$$

The penalties refer to the above mentioned pitfalls that may induce the GA to get stuck at local optima and *k* is a constant value that is explained later in this section. There are two penalty sources in the evaluation function:

- For each pair of repeated genes in a chromosome, the penalty term is increased by one unit, i.e., the fitness evaluation function drives the GA to achieve chromosomes with non-repeated genes.
- Genes that produce an output "1" where the truth table imposes "0" also contribute to the penalty term. The reason is that, even though this kind of gene may present a high number of hits, it should never take part in the final solution.[4]

We have arrived at these conditions through the experimental observation that, when the fitness evaluation function considered only the number of hits, the fittest

chromosomes were usually composed of many repeated genes, and some of them were not viable ones, as expressed by the second condition above. This kind of gene scored a good number of hits, but could not take part in the final solution. Figure 6.7 illustrates this drawback with an example.

a b c	ab	$\bar{a}bc$	a	ab + $\bar{a}bc$(Target)
0 0 0	0	0	0	0
0 0 1	0	0	0	0
0 1 0	0	0	0	0
0 1 1	0	0	0	0
1 0 0	0	0	1	0
1 0 1	0	1	1	1
1 1 0	1	0	1	1
1 1 1	1	0	1	1
Hits	7	6	7	

m1 = ab
m2 = $\bar{a}bc$
m3 = a

Figure 6.7 Example of the local optima problem that occurs when the fitness evaluation function takes only the number of hits into account.

The above figure shows a hypothetical Boolean function of 3 inputs and 2^3 cases in its truth table. The target function is optimally satisfied by the terms *m1* and *m2*, whose hits equal 7 and 6 respectively. Nonetheless, the term *m3* also scores a good number of hits, 7 out of 8; hence, it is more likely to be found by the GA comparing to *m2*. As we are using the sum of products paradigm, *m3* can not be part of the solution, as it will insert a 1 output in the case 100 (line 5), where the truth table specifies 0. This term will then be penalized by the new fitness evaluation function described in equation (6.2).

The term k in equation (6.2) is a constant set by the user, which will determine the strength of the penalty term in the fitness evaluation function. Through much experimentation, the authors have found best results for values of k around 2^n where n is the number of input variables. This can be explained by the fact that, in this case, both terms of the fitness function, hits and penalties, will have approximately the same range.

Gate Level

We observe the same problem when evaluating digital circuits represented at the

gate level, i.e., assessing only the hits at the circuit output is a misleading approach. The means whereby we can solve this problem is slightly different than the one described above: *reverse logic* is utilized in this case. This is accomplished by probing not only the circuit output point, but also internal circuit points. Figure 6.8 depicts this new evaluation procedure. In this figure, in addition to the circuit output, two internal points are also probed, and these three measurements are aggregated into the fitness evaluation function.

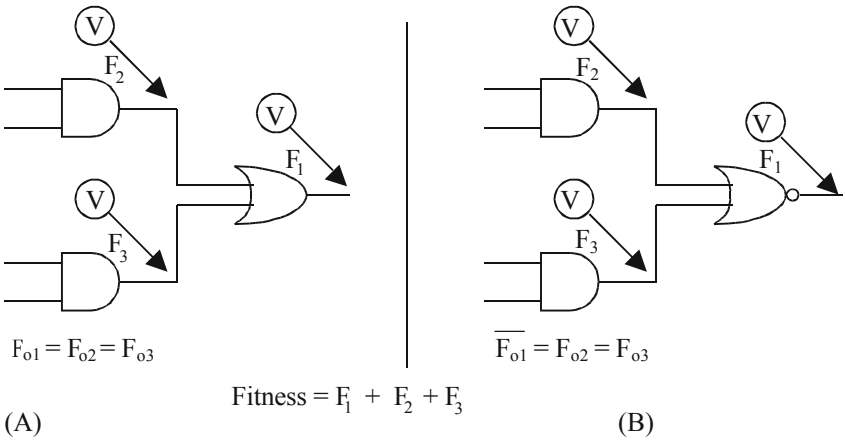


Figure 6.8 Evaluation method for the gate level representation. OR paradigm in (A), NOR paradigm in (B); F_i and F_{oi} respectively represent the output of the observed and the target values related to the point i. The V circles above represent points of voltage measurements.

We need therefore to address two questions related to this method. First of all, how can we assess the fitness of internal circuits' points (F_2 and F_3 in the above figure)? We remember that the fitness of the circuit output, F_1, is easily evaluated by a comparison with the specified truth table. As illustrated in Figure 6.8, we develop two evaluation paradigms to solve this problem, the OR (Circuit A) and NOR (Circuit B) paradigms. According to these paradigms, the output gate is respectively fixed to implement OR and NOR functions. By proceeding this way, the target internal points fitness (F_{o2} and F_{o3}) are the same as the target circuit output fitness (F_{o1}) for the OR paradigm, since the OR gate implements a logic sum; and they are the complemented value of the target circuit output fitness for the NOR paradigm, since the NOR function implements a complemented sum. Of course, we could also apply other paradigms, such as NAND, AND, and XOR, but the application of the reverse logic would not be so simple.

The second, and more important question is why does this procedure work? (Assuming that it works!). We do not have a strict proof of the advantage of evaluating internal circuit points, but we can provide convincing verbal arguments:

1. Reducing the search space: When we fix the output gate, we are limiting the search space to those possible solutions with the imposed gate at the output.

2. Discarding fit, but unfeasible solutions: By evaluating internal points, we can identify sub-circuits that will prevent the overall circuit from achieving the solution, as fit as they may be. In the case of the OR paradigm, just like in the *sum of products* representation, unfeasible sub-circuits are the ones that put a *1* in the associated internal point, when the truth table states *0*. Similar conditions may be found when using the NOR paradigm. Circuits presenting these conditions will be heavily penalized.

The case study section of this chapter provides an application example of this method.

Transistor Level

We said before that handwritten simulators were usually adequate to evaluate digital circuits. This applies to the two former representations, but not to the transistor level representation. This results from the fact that we are now evolving a digital behavior through analog devices, falling again in the case of analog circuit simulation, whose complexity requires standard simulators, such as *SMASH* and *SPICE*. Particularly, we will use these simulators in the *transient* mode.

The evaluation is accomplished in the following way: for a circuit with n inputs, the total period of the transient analysis, T, is divided into M slices of time. Since there are 2^n different cases in the truth table, the simulation will encompass $M = 2^n$ time slices that lasts $T/2^n$ units of time. Each time slice will present a target output voltage, either 0 Volt or 5 Volts, according to the correspondent case of the truth table. The fitness equation is then given by:

$$Fitness = -\sum_{i=1}^{M} | D(i) - R(i) |$$

This equation simply computes the sum of the absolute deviations between the desired voltage, D, and real output voltages, R, over M fitness cases. The negative sign is used because the fitness should be higher for smaller deviations.

6.3 CASE STUDIES

6.3.1 Combinational Circuits

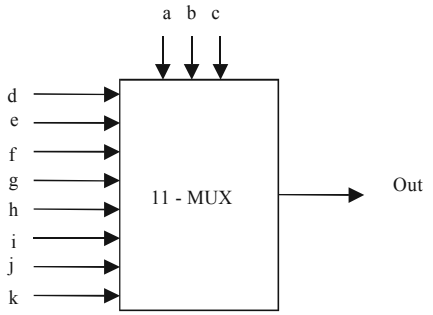
In this section we describe two sets of experiments, the evolution of multiplexers and parity functions, and the evolution of arithmetic circuits.

6.3.1.1 Multiplexers and Parity Circuits

The evolution of multiplexers and parity circuits has been commonly used to assess the performance of evolutionary algorithms.[11] These applications are not intended to discover new or more efficient designs, but to evaluate the potential of

evolutionary algorithms to *rediscover* human made designs.

We describe here the evolution of an 11-input multiplexer through the use of the *sum of products* representation. This device encompasses three selection lines and eight data lines, as shown in Figure 6.9.



a	b	c	out
0	0	0	d
0	0	1	e
0	1	0	f
0	1	1	g
1	0	0	h
1	0	1	i
1	1	0	j
1	1	1	k

Figure 6.9 Block diagram and truth table of the 11-input multiplexer.

The 11-input multiplexer is defined by 1,024 minterms* that are simplified to only eight terms. By adopting the sum of products representation, each gene encodes a product term in the form (a, b, c, d, e, f, g, h, i, j, k), where the literals are defined in the above figure. As described previously, each gene locus may assume three different values, 0, 1, and 2, referring to the state in which the correspondent literal appears in the product term. This problem presents many different solutions; however, there is one optimal solution, which encompasses the following eight genes:

(0,0,0,1,2,2,2,2,2,2,2)
(1,0,0,2,1,2,2,2,2,2,2)
(0,1,0,2,2,1,2,2,2,2,2)
(1,1,0,2,2,2,1,2,2,2,2)
(0,0,1,2,2,2,2,1,2,2,2)
(1,0,1,2,2,2,2,2,1,2,2)
(0,1,1,2,2,2,2,2,2,1,2)
(1,1,1,2,2,2,2,2,2,2,1)

There are 3^{11} = 177,147 different genes according to our representation. The complexity of picking up the 8 minimal terms from the total number of genes can be estimated as the combination of 8 over 177,147, $C_{177,147}^{8}$, which is approximately 2.41 x 10^{37}. So, we have a multi-modal task involving a vast search space, which precludes the use of standard evolutionary algorithms without enhancements. We can also note that the eight terms shown above present the same order (given by the number of *2s*) and, as a consequence, the same individual fitness.

* The number of minterms is equal to the number of cases in the truth table presenting a "1" output.

In order to tackle this problem, we have carved up uniformly the genome space into five slices, using, thus, five parallel GAs. The uniform decomposition of the search space is carried out by inducing each parallel running GA to sample genes only in a particular "slice" of the genome space, according to a gene indexing (Chapter 3, section 3.3). The first GA searches through the genes indexed from 0 to (177,147/5), the second searches from (177,147/5) to 2.(177,147/5), and so on.

Each GA processes 30 individuals along 150 generations. The memory paradigm and the ILG strategy (Chapter 3, section 3.1) have also been used. The genes found by each parallel GA may compose the overall solution, since they account for non-overlapping regions of the genome space and there is no epistatic effect in our representation. Using these criteria, 6 over 10 executions of the parallel GA found a solution; *four* of them found the optimal solution, and the other two achieved sub-optimal solutions. The sub-optimal solutions do not include the 8 genes mentioned above, solving the problem using more than 8 terms. Figure 6.10 shows the average fitness of the five GAs over ten executions; the best possible fitness is $2^{11} = 2{,}048$, achieved by none of the processes, since the solution is now distributed.

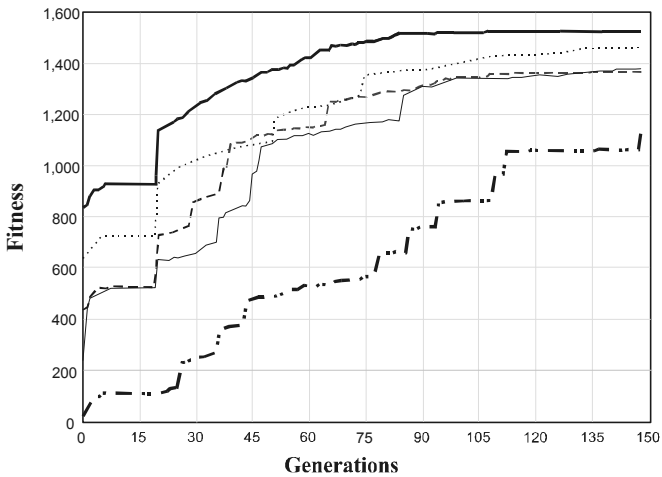


Figure 6.10 Fitness of the five GAs along the generations for the 11-MUX evolution.

We now present the application of this method to the evolution of another standard combinational function, the parity detectors. Particularly, we focus on the 8 inputs odd parity function, whose truth table produces an output "*1*" whenever there is an odd number of *1s* at the inputs. According to our representation, the search space is made up of 3^8 or 6561 genes. Contrasting with the multiplexer, there is, in this case, only one solution, which consists of 128 genes of the form:

(1,0,0,0,0,0,0,0)
(0,1,0,0,0,0,0,0)
(1,1,1,0,0,0,0,0)
.
.
.

and so on. All the 128 genes are *non-simplifiable* minterms, since all the input variables are present (no *2* symbols). We can then estimate a complexity of C_{6561}^{128}, approximately equal to 10^{271}. In order to tackle this problem, the genome space has also been divided, in this case into ten regions, with one GA used to sample each region.

Figure 6.11 shows the best individual fitness along evolution for each GA; the ones that sample regions with more minterms reach higher fitness values. At the end, all 128 minterms have been found. Nevertheless, this is an example for which our representation is not efficient: around 10^6 individuals were sampled by the GAs, requiring too much computation time and resources. This example exposes one drawback of the sum of products representation: parity functions can be more easily expressed using the exclusive OR operator that is not used in this representation.

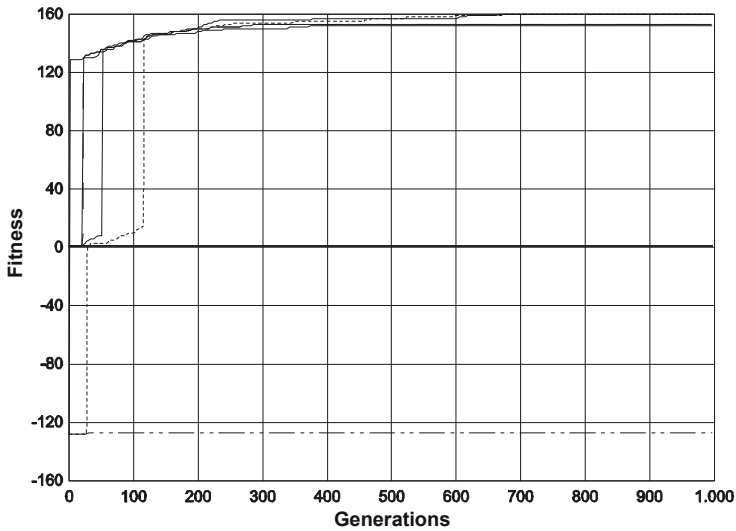


Figure 6.11 8-parity experiment: fitness along 1000 generations for the 10 GAs. Each line corresponds to one GA; note that five of them show a zero fitness throughout the evolution, because there is no solution in the "slice" of the search space sampled by them.

6.3.1.2 Arithmetic Circuits

Arithmetic circuits are used in the Arithmetic Logic Unity (ALU) of standard microprocessors, as well as in Digital Signal Processors (DSPs). This section

describes examples of the synthesis of comparators, adders, and multipliers. Two different representations are used, the sum of products and the gate level representation.

6.3.1.2.1 Sum of Products Representation

This section has a twofold purpose: to show the use of the sum of products representation in the synthesis of a digital comparator; and to provide the reader with a more quantitative analysis of the efficiency of the genetic memory technique, described in the third chapter of this book.

We describe the evolution of a comparator that implements the logic function A > B, where A and B are 4-bit numbers, resulting in a total of eight inputs. The truth table of this combinational function contains 120 minterms, which may be simplified to the following 15 terms presented in Table 6.1:

Table 6.1 Minimal solution achieved by Espresso for the 8-bit comparator. It comprises 15 terms, denoted by *t1*, *t2, ..., t15* defined according to our chromosome representation; A = a3a2a1a0 and B=b3b2b1b0.

	a3	**a2**	**a1**	**a0**	**b3**	**b2**	**b1**	**b0**
t1	2	2	2	1	0	0	0	0
t2	2	2	1	2	0	0	0	2
t3	2	2	1	1	0	0	2	0
t4	2	1	2	2	0	0	2	2
t5	2	1	2	1	0	2	0	0
t6	2	1	1	2	0	2	0	2
t7	2	1	1	1	0	2	2	0
t8	2	1	1	1	0	2	2	0
t9	1	2	2	2	0	2	2	2
t10	1	2	2	1	2	0	0	0
t11	1	2	1	2	2	0	0	2
t12	1	1	2	2	2	0	2	2
t13	1	1	2	1	2	2	0	0
t14	1	1	1	2	2	2	0	2
t15	1	1	1	1	2	2	2	0

By the genes shown above, it can be observed that, whereas the solution of the multiplexer function is made up of high order terms (highly simplified terms) and the solution for the parity function is composed of the lowest order terms (the own minterms), the solution of the 8-bit comparator includes both high order terms (t9) and low order terms (t1, t7, t8, and others with a small number of 2s in Table 6.1). There are a total of $3^8 = 6561$ possible genes. The complexity of gathering 15 terms can be calculated by C_{6561}^{15} which is approximately equal to 10^{45}.

We have already investigated the use of the sum of products representation in the case of multiplexers and parity detector circuits. In these examples, due to the size of the search space, we had to include in the GA some techniques to improve the performance, such as the memory paradigm and the decomposition of the search

space. We will now focus only on the use of the genetic memory. The effectiveness of the re-use of genetic material, when initializing non-coding segments along evolution, may be demonstrated by the synthesis of this function (8-bits comparator). We have also adopted Variable Length Representation in this experiment. As explained in Chapter 3, non-coding segments arise from the use of variable length representation, being characterized by the genes that do not contribute to the chromosome decoding process (inactive genes). These genes are gradually activated along the evolutionary process.

Table 6.2 shows the average performance of our methodology when using the OLG (Oscillating Length Genotypes) and the ILG (Increasing Length Genotypes) strategies (Chapter 3), both with the memory paradigm, and the OLG strategy without the memory paradigm. In this table, *% of success* indicates the percentage of GA executions that achieved a correct solution; G_{av} and F_{av} are, respectively, the size and fitness of the best genotypes achieved, averaged over ten executions. As it can be seen, the re-use of genetic material in non-coding regions produces an extraordinary benefit to the performance in this case. It can also be observed that the ILG strategy obtained more parsimonious solutions in terms of number of genes; the best solution found presented a total of 16 genes, one gene more than the optimal solution. This result agrees with the hypothesis stated in Chapter 3, about the potential advantages of the ILG strategy: by starting sampling small chromosomes and gradually increasing their sizes, there are more chances for the GA to converge to more parsimonious solutions. The strategies employing the memory paradigm took around four hours to run 10 executions in a SPARC Classic workstation.

Table 6.2 Evolution of the 8-bits comparator using the sum of products representation.

Methodology	**% of Success**	**G_{av}**	**F_{av}**
OLG with memory	100	22.1	256
ILG with memory	80	18.5	255.8
OLG without memory	0	8.1	243.8

6.3.1.2.2 Use of Reverse Logic and Incremental Evolution

The sum of products representation, although adequate to handle combinational functions reaching 11 inputs, does not have the potential to find novel digital circuits, which could outperform conventional design. This fact stimulated the authors to migrate to the gate level representation. Particularly, we address again the 8-bit comparator through this representation.

The main problem of the gate level representation is its application to the evolution of larger digital circuits, typically those with more than six inputs. In the particular

case of the 8-bit comparator, a combination of the evaluation using reverse logic and incremental evolution has been employed to overcome this problem. The evaluation in reverse logic has already been described in this chapter, consisting of the OR and NOR paradigms. Incremental evolution is another procedure that can be generally applied to the solution of complex problems. This procedure is summarized by the architecture shown in Figure 6.12.

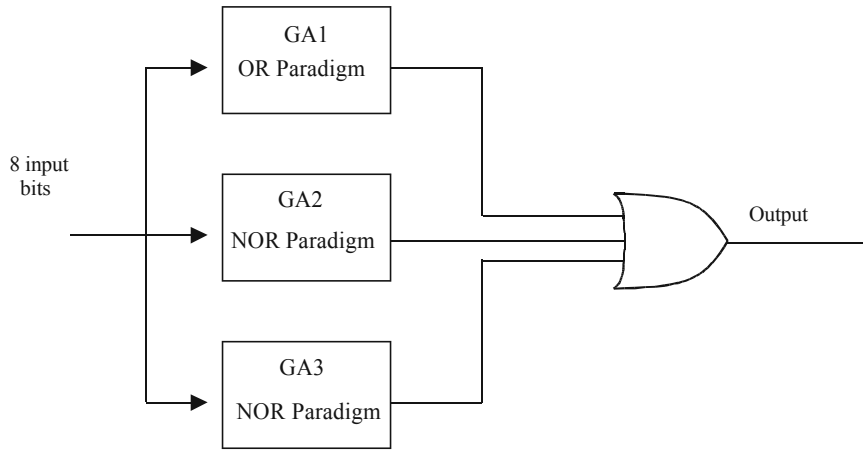


Figure 6.12 Architecture employed in the evolution of the 8-bits comparator using the gate level representation.

We can see, through the architecture depicted above, that three GAs have been utilized to solve this problem. However, when using incremental evolution, one does not know a priori how many GAs will be required to completely solve the problem. The first step was to employ a GA that manipulated a cell like the one in Figure 6.5, but without the flip-flop. The OR and NOR evaluation paradigms have been tested, where the best results have been achieved with the OR paradigm (GA1 in Figure 6.12). However, this execution could not provide a solution that thoroughly satisfied the 8-bit comparator truth table. Here starts the incremental evolution *trick*: we store the first evolved circuit and launch another GA, considering now another truth table, where the elements already solved by the first circuit are not taken into account. Again, the OR and NOR paradigms are tested, with the latter presenting better results (GA2 in Figure 6.12). The process is then repeated until the truth table is fully satisfied. In this particular case, the problem was solved through three GAs, one of them using the OR paradigm and the other two using the NOR paradigm. The overall circuit is depicted in Figure 6.13.

Before drawing comments on the above circuit, let us make an important remark on this procedure. As shown in Figure 6.12, the sub-circuits evolved by the different GAs are connected by an OR gate. This works well if these partially functional circuits correctly address all the cases where the truth table has a "0" output – they only fail by delivering "0" in cases where the correct output is "1." As we have already explained, this is a consequence of the use of an OR gate at the output.

The circuit depicted in Figure 6.13 used 47 two-input logic gates. If the design

was to be accomplished by Espresso, encompassing the product terms shown in Table 6.1, and implemented using only two-input logic gates, 63 gates would have to be used. Therefore, assuming the technological constraint of using only two-inputs gates (observed, for instance, in FPGAs), the evolved design is more efficient compared to the conventional design. The superiority of the GA evolved cell is not only a consequence of the two-input logic gates constraint, but it is mainly due to the fact that the GA manipulates XOR logic together with AND-OR logic, whereas the software Espresso only manipulates the latter.

This result suggests that evolutionary systems are able to outperform conventional tools for digital design when technological constraints are considered, and, more importantly, when the XOR logic may be used together with the AND-OR logic. The reason is that human conceived strategies to digital design do not contemplate the above conditions. This conclusion is reinforced by the results shown in the next section.

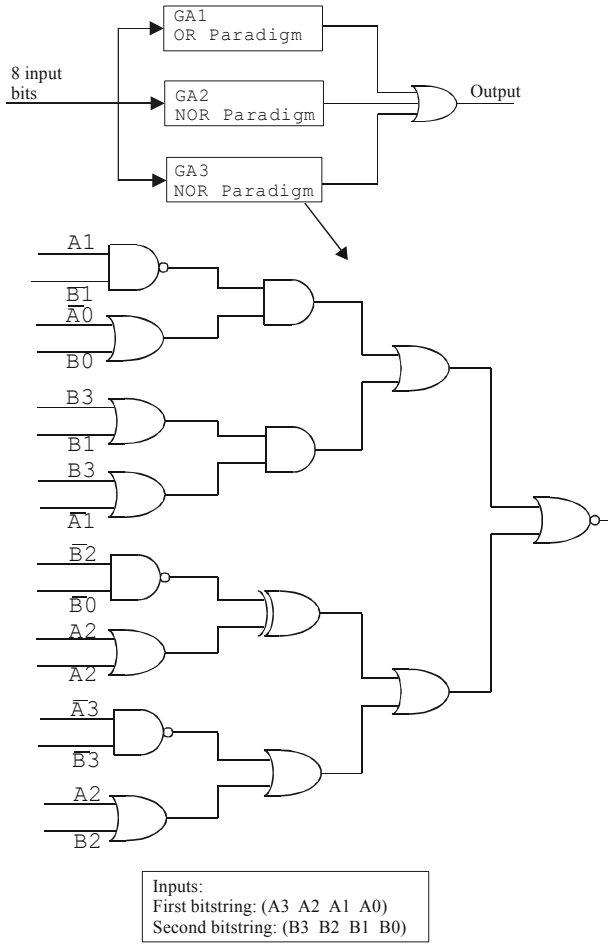


Figure 6.13 8-bit comparator evolved through the gate level representation.

6.3.1.2.3 Adders and Multipliers

Miller[10] et al. (1997) described a set of experiments where they accomplished the evolution of adders and multipliers. This work demonstrated that evolutionary systems were not only able to rediscover standard designs for these arithmetic circuits, but also, in some cases, to improve them.

They employed a gate level representation based on the Xilinx XC6216 Field Programmable Gate Array (FPGA) architecture. As it will be described in the next chapter, each cell of this FPGA can implement, among other functions, the AND, NAND, OR, NOR, XOR, and MUX logic functions. Their gate level representation is similar to the one used previously, as described:

> *There are two aspects necessary to define any combinational logic network. The first is the cell-level functionality and the second is the inter-connectivity of the cells between circuit inputs and outputs. A chromosome is defined as a set of interconnections and gate level functionality for these cells from output back toward inputs based upon a numbered rectangular grid of the cells themselves, as in (Figure 6.14). This procedure was carried out in such a way that all individual cell inputs could only be connected to cell outputs which were of a lower number within this scheme. This is important because it eliminates any possibility of feedback connections which would give rise to non-combinational time-dependent behaviour. the inputs that are made available are logic '0', logic '1', all primary inputs and primary inputs inverted. To illustrate this, consider a 3x3 array of logic cells between two required primary inputs and two required outputs.*

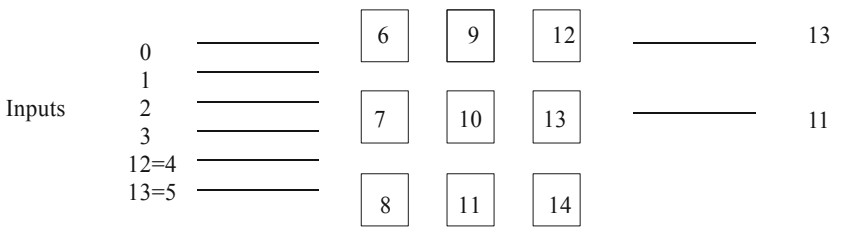


Figure 6.14 Logic cells arrangement in the gate level representation proposed by Miller. (Extracted from Miller[10] et al. (1997))

> *The inputs 0 and 1 are standard within the chromosome, and represent the fixed values, logic '0' and '1' respectively. The inputs (two in this case) are numbered 2 and 3, with 2 being the most significant. The lines 4 and 5 represent the inverted inputs*

> *2 and 3 respectively. The logic cells which form the array are numbered column-wise from 6 to 14. The outputs are numbered 13 and 11, meaning that the most significant output is connected to the output of cell 13 and the least significant output is connected to the output of cell 11. These integer values, whilst denoting the physical location of each input, cell or output within the structure, now also represent connections or routes between the various points. In other words, this numbering system may be used to define a netlist for any combinational circuit. Thus, a chromosome is merely a list of these integer values, where the position on the list tells us the cell or output which is being referred to, while the value tells us the connection (of cell or input) to which that point is connected, and the cells functionality.*

We can contrast this gate level representation with the one previously employed in the evolution of the 8-inputs comparator:

- In both representations, care was taken to avoid feedback connections.
- Besides the basic logic functions, Miller's representation also allows a cell to implement a 2-1 multiplexer function.
- In terms of geometry, Miller's representation forms a rectangular array of logic cells, while, in the previous representation, the logic cells are arranged in a triangular array.

The fitness evaluation function used was a very simple one, considering only the percentage of total correct outputs in relation to a truth table.

Initially, they performed an extensive set of experiments in order to find out acceptable values for population size and number of generations. As they remark:

> *Initially large populations were chosen (between 1000 and 2000). However it was later discovered that relatively small populations performed better. The breeding rate, defined as the percentage of population subjected to crossover and replaced by children, was found to be best set at 100%. The percentage of all genes in the population which were mutated before breeding was chosen to be 5%. Experiments confirmed this as a good figure. Contrary to initial expectations it was found to be beneficial to employ elitism. Uniform crossover was used.*
>
> *It was clear ... that a population size of between 15 and 60 with a large number of generations (between 40,000 and 80,000)*

performs best, and quite markedly better than the larger populations.

Using this framework, they achieved very interesting arithmetical circuits. Figure 6.15 depicts a two-bit full adder with carry, evolved in an experiment that included 20 different executions that processed 50 individuals along 50,000 generations. Opposite to most of the circuits shown in this book, this two-bit full adder demonstrates the evolution of a hierarchical design, since the sub-circuit that handles the least significant bit is identical to the one of the most significant bit.

Figure 6.16 depicts a two-bit multiplier, whose inputs are A1, A2, B1, and B2, and the outputs are P1, P2, P3, and P4. This circuit was obtained after 50 runs of the GA with a population size of 80, each run terminating at 60,000 generations. As the authors remark, this two-bit multiplier used only seven gates, being more efficient than human made designs (that require 8 gates).

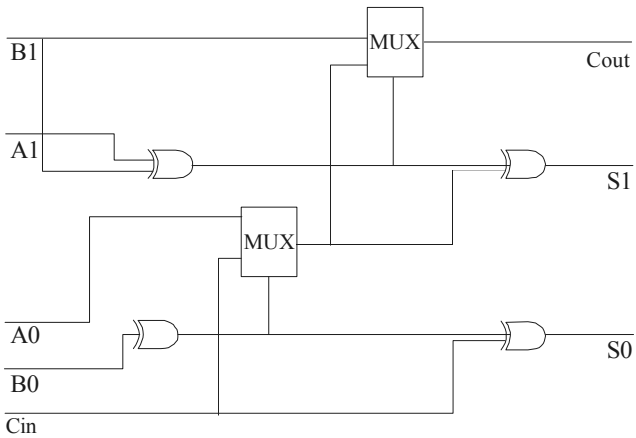


Figure 6.15 Evolved two-bit full adder with carry reported in Miller et al. (1997).

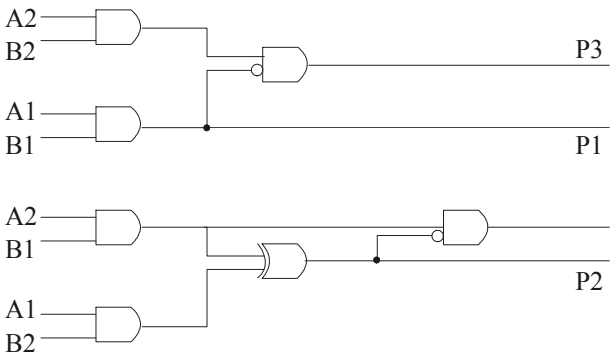


Figure 6.16 Evolved two-bit multiplier (Figure 6.13). (Extracted from Miller.[10])

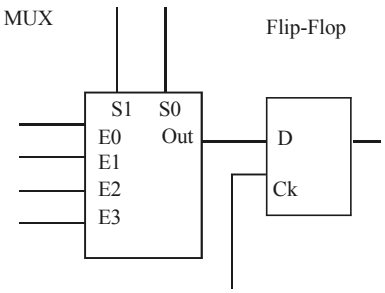
We can compare the results of the 8-bit comparator obtained using the reverse logic evaluation, described in the last section, and the results achieved by Miller[10] et al.:

- Both of them use the gate level representation and can be implemented in some FPGAs models.
- While most of Miller's experiments focus on circuits with up to 6 inputs, but with multiple outputs, the comparator experiment involved 8 inputs, but a single output.
- 100% compliant circuits were obtained in both experiments.
- Both experiments achieved circuits more efficient, in terms of number of gates, than the ones conventionally designed.
- While Miller's experiments involved the sampling of around 10^7 individuals, the use of incremental evolution with reverse logic evaluation allowed the solution to be found after processing around 10^6 individuals in the case of the comparator.

6.3.2 Synthesis of Sequential Circuits

The investigation of the evolutionary synthesis of sequential systems is less advanced in comparison with their combinational counterparts. Nevertheless, since the use of feedback provides a digital system with a wider repertoire of dynamics, evolutionary systems will possibly bring more remarkable results in this area.

We describe an experiment in which a sequence detecting circuit must be evolved. Particularly, this circuit must identify the four bits sequence *0 1 0 1* out of a word of 16 bits. Once one sequence is detected, the circuit output must switch from "0" to "1," and stay in this state until the end of the 16 bits word. This is a typical introductory problem of sequential logic.[4] In order to solve this problem, we first tried the representation depicted in Figure 6.5 with poor results. We then tried a similar gate level representation based on multiplexer logic: instead of using an arrangement of logic gates, a multiplexer is the combinational part of the basic cell. Figure 6.17 depicts the cell. The chromosome is made up of genes, each of which encodes multiplexer's inputs (S1, S0, E0, E1, E2, and E3) of a particular cell. The inputs are selected from the external input (corresponding to the sequence input line) and from feedback signals (flip-flop outputs of the same cell and of other cells).



Gene=[I(EO), I(E1), I(E2), I(E3), I(S0), I(S1)]

I implements the mapping between each multiplexer input and the signal that should be applied to it

Figure 6.17 Multiplexer-based cell utilized to represent circuits in the experiment of the sequence detector.

Each gene includes six loci, each one selecting the input of a particular multiplexer input. In this particular experiment, we employed a fixed length representation: each chromosome is composed of four genes. Therefore, the resultant circuitry will include four cells like the one shown above. The flip-flop clocks are all fed by the same master clock line, with a positive edge at the center of the sequence bits.

Theoretically, we should evaluate each circuit with 2^{16} or 65536 words, covering all the possible cases. In order to avoid this extremely time-consuming procedure, we selected only 11 words by hand, some of them including the target sequence and some of them not. A circuit has been evolved which was able to answer correctly to the 11 words, and also to generalize and answer correctly to the overall 65536 words. Figure 6.18 depicts this circuit, obtained after 20 executions that processed 40 individuals along 400 generations.

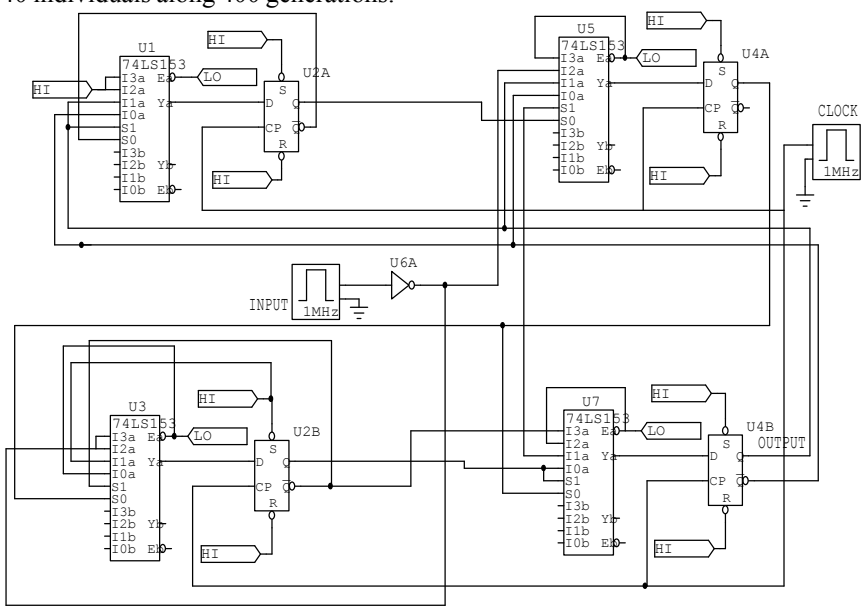


Figure 6.18 Schematic (in PSPICE editor) of the sequence detecting circuit achieved by the GA.

We can draw a number of conclusions regarding the result achieved by the GA:

- The fact that the evolved circuit was capable to generalize from 11 words to 65536 words was very impressive.
- This automatic design uses less amount of human knowledge than conventional tools for sequential circuits design. The latter usually requires a state machine of the circuit as an input. Our system learns through examples and does not require this kind of information.
- We did not choose 11 random examples out of 65536. Some human knowledge was needed to select meaningful words: some of them including the target sequences in different points of the word, some of them including sequences similar but not equal to the target one, etc.
- This automatic design is not the most efficient in terms of amount of hardware. This system can be specified by a machine with five states that can be realized using only three flip-flops. The circuit shown above uses four flip-flops.

6.3.3 Transistor Logic

This set of experiments involves the synthesis of two-inputs OR, AND, and XNOR gates using the transistor level representation. Before the advent of CMOS logic, the use of transistor logic was widespread in ICs implementing logic gates. Although, nowadays, CMOS technology is more common in this kind of ICs, bipolar transistor logic is still more efficient in terms of speed.

In the case of the evolution of the OR gate, we have utilized a GA with variable length representation and ILG sweeping strategy. The population was initialized with chromosomes presenting four active genes, being able to reach up to eight active genes. Figure 6.19 depicts the schematic of the evolved OR gate; Figure 6.20 presents the circuit response in the time domain; finally Figure 6.21 plots the average chromosome length of the population over 10 executions. Forty individuals have been processed through 30 generations in each execution.

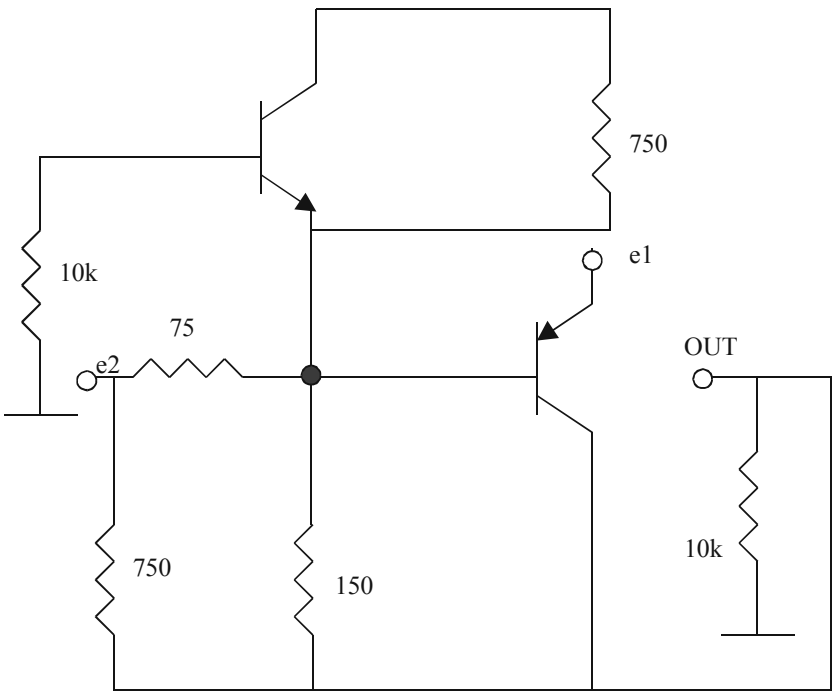


Figure 6.19 Schematic of the OR gate synthesized by the Genetic Algorithm.

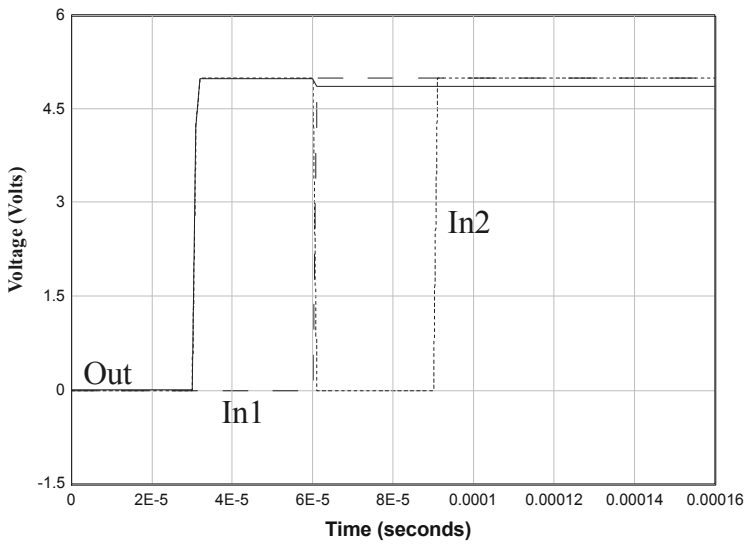


Figure 6.20 Time domain response of the circuit shown in Figure 6.19. Inputs are labeled *In1* and *In2*. Output is labeled *Out*.

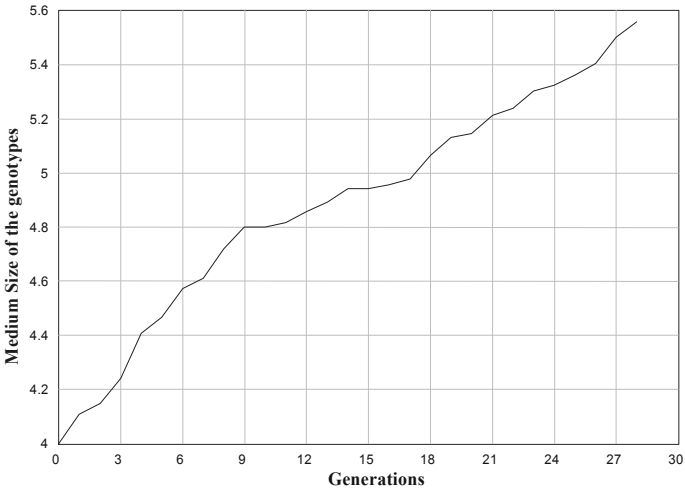


Figure 6.21 Average chromosome size in the evolution of the OR gate.

The circuit depicted in Figure 6.19 implements the function *OUT = e1 OR e2*, according to the definitions given to the input and output points. A load resistor of 10k was employed. The circuit's behavior was verified using discrete components. This circuit outputs 0 Volts in case of "0" logic level and a minimum of 4.87 Volts in case of "1" logic level. Similarly to the procedure described in the evolution of analog amplifiers, a penalty term has been incorporated into the fitness evaluation function to eliminate unfeasible circuits, the ones with transistors in over-voltage or over-current conditions. The evolution of the AND gate has been accomplished in a similar way.

The synthesis of a XNOR gate is a more interesting case study to be tackled by a GA. The XNOR function belongs to the class of the non-linearly separable Boolean functions, consisting of a non-trivial case to the GA. Figure 6.22 depicts the schematic of the XNOR gate and Figure 6.23 displays its time response. In this case, a fixed-length representation GA was used. This circuit was achieved in an experiment involving the sampling of 50 individuals along 50 generations. This circuit was also implemented through discrete components, supporting the result obtained in simulation.

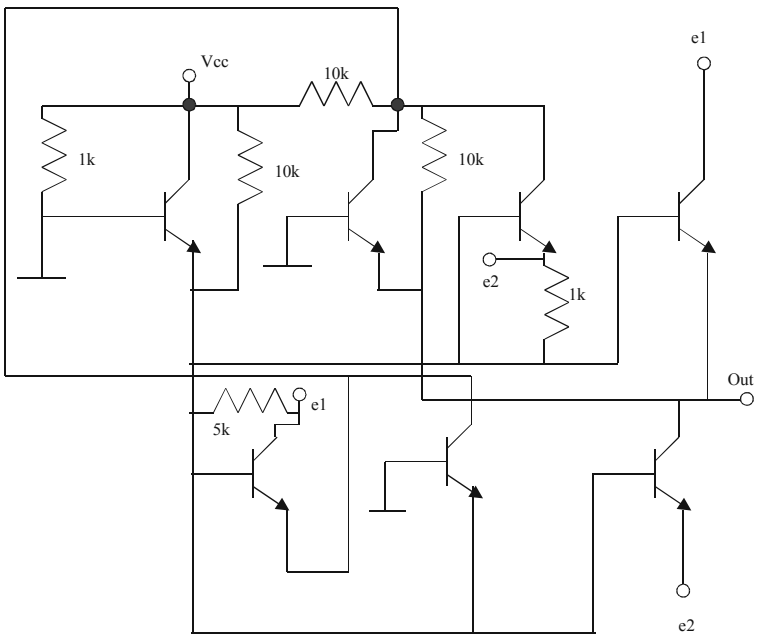


Figure 6.22 XNOR gate synthesized by the GA (Output = Out, Inputs = e1 and e2).

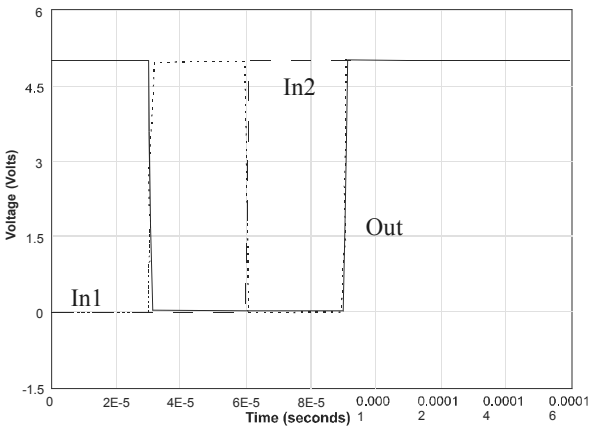


Figure 6.23 Time domain response of the circuit shown in Figure 6.22. Inputs are labeled *In1* and *In2*. Output is labeled *Out*.

This section has just introduced a promising application of , and there is much to be done. We can not yet establish a fair comparison between these evolved circuits and the human designed ones, for instance, in the context of the Transistor-Transistor-Logic (TTL). Human designed projects consider many practical issues such as:

- Boundary values for the logic levels or noise margin: $V_{oh,mín}$ minimum voltage level accepted as logic "1"; and $V_{oL,máx}$, maximum voltage level accepted as logic level "0".

- Dissipation.

- Load capacity or fan-out: how many logic gates can be connected to the output of the evolved gate.

Depending on the particular application, there may be other aspects to be considered. The authors feel encouraged by the fact that good multi-objective methodologies may cope with this multi-criteria kind of design. Thus, incorporating more objectives into the GA is the next step.

The authors also foresee the evolution of more complex gates as a potential outcome of these experiments. Here, we feel tempted by the fact that rather innovative circuits may appear. For instance, a 6-input multiplexer is typically made up of a well-known arrangement of AND and OR gates, because the human designer employs a gate-level abstraction when dealing with more complex designs. The GA has, in principle, the potential to handle this or other cases at the transistor level. However, as we increase the number of inputs, the GA performance also suffers due to the increase in the inherent difficulty of the problem. New GA models will have to be investigated to handle these cases: speciation and genetic memory are possible ways to overcome this difficulty.

6.3.4 Digital Filters

We present here the last example of evolutionary digital design. There are of course other case studies on digital design that can be tackled by evolutionary systems, but the four cases presented in this chapter cover a wide range of applications.

We have demonstrated that artificial evolution was able to improve human design in the case of combinational circuits. However, this was achieved either through a computational effort that involved the sampling of a large number of individuals, or by devising new evaluation techniques, such as the reverse logic. Evolutionary Computation can be successfully applied to obtain small digital cells that outperform human designed ones, but the challenge of evolving more complex digital systems still remains.

The area of digital filters is the most promising for the evolutionary design of complex digital systems. This section is then dedicated to this type of circuit, comprising the following topics. We start by introducing the basic concepts of digital filters; next, we contrast two evolutionary approaches to the automatic synthesis of digital filters – the *functional level* and the *gate level*. Finally, we provide application examples of both evolutionary approaches.

6.3.4.1 Basic Concepts

Digital filters can be divided into two classes: Finite Impulse Response (FIR) and Infinite Impulse Response (IIR). Both classes can be described by difference equations:

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) + \sum_{i=1}^{M} b_i y(n-i)$$

The filter output, y(n), is a function of the previous inputs, x(n–i), and of the previous outputs, y(n–i). In the case of FIR filters, only the first term is used, whereas IIR filters also use the output feedback. The filters' coefficients, a_i and b_i, are real numbers. Interested readers can refer to Eshelman and Schaffer[12] (1989) for a complete description of digital filters theory.

The difference equations of the digital filters can be either implemented by a general purpose microprocessors, or by a DSP (Digital Signal Processor) for more efficient implementation. DSPs are implemented through shift registers, adders, and multipliers, which are the necessary hardware to implement the difference equation. The signal x(i) is originally analog; so it, should be sampled and converted to the digital domain beforehand. After the filtering action takes place, the output y(i) is converted back to the analog domain. This scheme is presented in Figure 6.24.

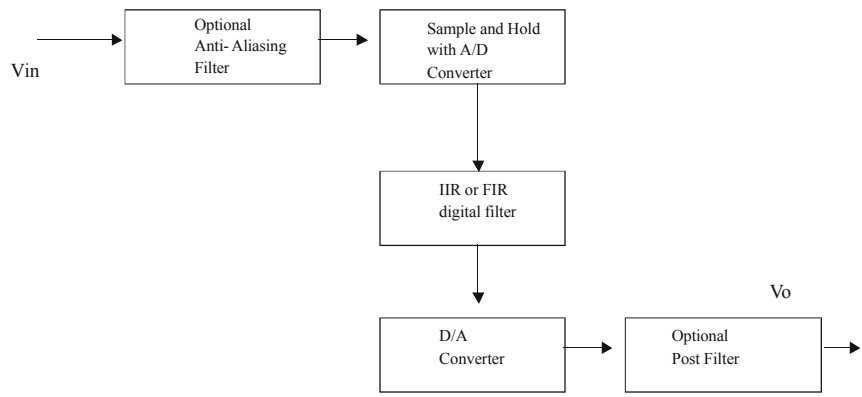


Figure 6.24 Block diagram of a digital filter and its interface with the analog world. (Extracted from Ellis, 1994.[16])

6.3.4.2 Functional and Gate Level Representation

The functional and gate level representations for digital filters evolution are well described in Miller[13] (1999):

Recently, researchers have started to explore the application

> *of evolutionary algorithms to filter design. The essential idea employed by most of these authors to use an evolutionary algorithm is to optimize the filter coefficients. This may be in combination with finite word length analysis for IIR filter design, or it may be in an adaptive context. Other workers have employed evolutionary algorithms to optimise coefficients together with add and shift operations in so-called multiplier-less designs.*
>
> *One of the objectives of this paper is to show that the possibilities afforded by gate-level evolution have been left largely unexplored, and that there remains fundamental work to be done at this level.*

So, while the functional level representation optimizes the filter coefficients, the gate level representation uses simple logic functions to implement the filter. We will show examples of both approaches and draw conclusions about the advantages and disadvantages of each one.

6.3.4.3 Functional Level Representation in the Synthesis of Multiplier-less Filters

In this representation, the chromosome encodes the filter's coefficients. Schaffer and Eshelman[14] (1993) proposed this representation to the design of multiplier-less FIR digital filters, the ones in which the coefficients are restricted to the form 2^n. In this particular class of filters, the multiplication operation can be simplified to a simple shift operation. However, this kind of filter also presents disadvantages, as explained by the authors of the referred work:

> *Unfortunately, coefficients sets limited to power of two tend to produce very limited range of frequency responses, but if several FIR filters are cascaded (i.e., the output of one is input to the next) then the coefficient set of the equivalent single-stage filter is simply the convolution of the coefficients from individual stages.*
>
> *To design a linear phase digital FIR filter using a genetic algorithm we must: (1) design a representation for coding power-of-two coefficients in a chromosome, (2) allow the user to specify the number of coefficients in each stage of a cascade, and (3) allow the user to specify the goal frequency response.*

In the reported application, each filter coefficient is represented by a four-bit sequence, following a Gray binary code. This encoding is reproduced in Table 6.3:

Table 6.3 Binary encoding of the filter coefficients. (Extracted from Schaffer and Eshelman[14] (1993))

Bits	Coefficient
0000	-1.0
0001	-0.5
0011	-0.25
0010	-0.125
0110	-0.0625
0111	-0.03125
0101	-0.015625
0100	0.0
1100	0.0
1101	0.015625
1111	0.03125
1110	0.0625
1010	0.125
1011	0.25
1001	0.5
1000	1.0

A variable length representation would be more adequate to handle this kind of problem, since one does not know a priori the number of coefficients of the filter (the filter length). Nevertheless, Schaffer and Eshelman chose to use a fixed length representation, where they estimate reasonable number of stages and of coefficients per stage for the filter (3 stages and around 5 coefficients per stage).

In the particular case of multiplier-less filters, after decoding the coefficients, their convolution must be implemented to determine the equivalent single-stage coefficient set. The filtering actions are then assessed by examining the impulse response of the single-stage filter, through the computation of the Fourier transform of the coefficient set using the Fast Fourier Transform Algorithm (FFT).

The goal frequency response was specified using lower and upper limits of the magnitude of the frequency response. For instance, they specified a band-pass filter in the following way:

Table 6.4 Band-pass filter specification. (Extracted from Schaffer and Eshelman[14])

Frequency (% Nyquist)	Lower Limit	Upper Limit	Weight
0.0	–9999.0	–30.0	1.0
10.0	–9999.0	–30.0	0.0
12.0	–9999.0	–3.0	0.0
20.0	–0.25	0.25	5.0
30.0	–0.25	0.25	0.0
37.0	–9999.0	–3.0	0.0
40.0	–9999.0	–30.0	1.0
100.0	–9999.0	–30.0	1.0

In the above table, a band-pass filter is specified with a passing band between 20 and 30 (measured in percent of the Nyquist frequency). The fitness function utilized

was then given by:

$$filter\ \_score = \max_{0 \le i \le 255} \{w_i \cdot rv_i\}$$

The filter score considers 256 points produced by the FFT. The function rv_i takes a minimum value when the filter output is precisely at the center of the specified envelope; it grows linearly when the filter output goes out of the specification. Naturally, the final fitness will be the inverse of the value given above. Schaffer and Eshelman describe the result for the band-pass filter:

> *After considerable trial and error (up to two days for one filter), the experts had found suitable PO2 filters and we began with the specifications and cascade structure they had used (i.e., we knew feasible solutions existed). A within-spec design emerged for the band-pass filter after about 10 minutes (on a Sun Sparc Station 1+) and repeated experiments yielded similar results every time. These experiments ran for an average of 137000 trials (about an hour) ...*

They used the number of logic gates as the metric to compare the GA results with human designed filters. They observed that, in some cases, the evolved digital filter used less logic gates than human designed ones. For instance, for a filter with band-pass ripple of 0.40 dB and a stop-band attenuation of –50.1dB, the evolved filter could be implemented with 8040 logic gates, against 13350 for the human designed filter.

Although the functional level representation can produce some benefits to digital filter design, there are some potential limitations:

- As only the filters' coefficients are evolved, implementation requirements are not taken into account.
- As implementation requirements are not considered, this methodology is not able to find novel circuits.
- There are many software packages available to optimize filters' coefficients, being a mature field of research.

When the GA manipulates only the filter's coefficients, the hardware implementation will always be constrained to the basic components' arrangement of DSPs, with shifters, adders, and possibly multipliers. Although it is clear that this implementation is efficient for digital filters specified by difference equations, it is not proved that this is the optimal implementation if the difference equation representation is not utilized. The gate level representation may be a more efficient

approach to this kind of unconstrained design.

6.3.4.3.1 Gate Level Representation

In contrast to the functional level representation for digital filters, the gate level mapping does not assume that the filter can be expressed through an analytical difference equation. Therefore, instead of optimizing the filter's coefficients, a direct synthesis of the digital circuitry that carry out the filtering task will be tried.

This is a very challenging problem, since a fine grained digital hardware, usually consisting of basic logic gates, will be used as basic building blocks for the evolutionary system. This contrasts with the conventional hardware used to implement digital filters, i.e., adders, shift registers, and multipliers.

This approach is also very tempting, since one can not say that the typical hardware implementation of difference equations, with or without optimized coefficient, is the best one for a digital filter. If we remove the constraint of achieving a filter that is mathematically expressed by an analytic equation, we can find novel circuits that carry out the filtering task. The main advantage of these new circuits is the possibility to have less amount of hardware used by them, since they are an arrangement of basic logic gates.

Miller[13] (1999) has used a gate level representation similar to the one previously described in his experiments targeting the evolution of arithmetic circuits. In this case, however, the only allowed gate function was a multiplexer. We will now list the main features of Miller's experiment:

- The input and output samples were digitized and represented as a word of eight bits.
- A gate array of seven rows and seven columns was used, meaning that each circuit included a total of 49 multiplexers.
- As he was interested in reproducing the behavior of FIR filters, the *n* last input samples were available to be used as multiplexers' inputs. In his case, *n* was set to 6, giving thereby a total of 48 (6 input samples x 8 bits) input bits to the gate array.
- Only feedforward connections were allowed in the gate array.
- He ran 25 experiments including sampling 5 individuals along 10000 generations.

Two different fitness evaluation functions have been used, one considering the frequency response through the computation of the DFT (Discrete Fourier Transform) and another one considering the circuit response in the time domain.

The frequency domain fitness evaluation function is defined by the following equation:

$$x_i^F = \delta_i \cdot (W(f_i) - W_{\max}^i) \cdot \frac{r_i}{n_+} + (1-\delta_i)(1-W_{\max})\frac{r_i}{n_-}$$

where x_i^F represents the fitness at the particular frequency *i*. d_i is 1 if the frequency *i* is to be passed, and 0 if it is to be stopped. *W(f)* represents the power in the frequency domain, defined as the modulus of the output response in the complex frequency domain. $W_{\max}$ is the maximum power over all frequencies:

$$W_{\max} = \max\{W(f_j), \forall j : f_1 \le f_j \le f_{n-1}\}$$

$W_{max}^{\ i}$ is the maximum power over all frequencies excluding f_i:

$$W_{\max}^i = \max\{W(f_j), \forall j \qquad j \ne i : f_1 \le f_j \le f_{n-1}\}$$

The value of r_i refers to a used defined set of fitness reward at the particular frequency *i*. Finally, n_+ and n_- represent, respectively, the number of frequencies to be passed and stopped.

The second fitness function does not carry out a DFT, but it is based on the difference between the actual circuit output and the desired output, being defined by the following equation:

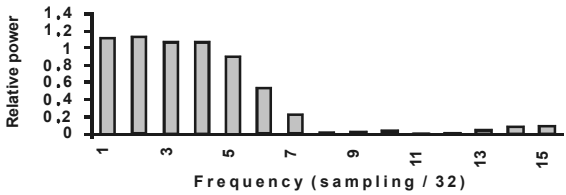
$$x_i^E = \left(\frac{\delta_i}{1+\frac{E}{K}}\right)\frac{r_i}{n_+} + \left(\frac{1-\delta_i}{1+\frac{Y}{K}}\right)\frac{r_i}{n_-}$$

In the above equation, one is trying to minimize the differences between the output and the input signals in the passing band region, and minimize the output signal in the stop band region. The error *E* is defined as the sum over samples of the absolute difference between the filter's input and output signals. Y is defined as the sum over output samples y_j, and *K* is a scaling constant. The total fitness x^F or x^E associated with a given chromosome is then given by the sum of components x_i for all frequencies up to f_{n-1}. The first fitness equation is called frequency-based fitness and the second is called error-based fitness.

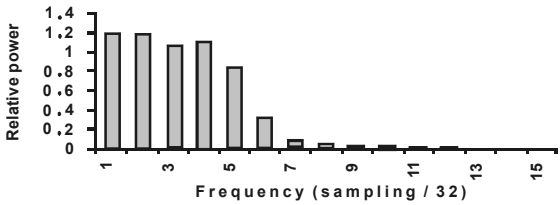
Miller evolved two low-pass filters, each using one of the above fitness equations. The frequency responses of the filters are shown in Figure 6.25. The passing-band spanned from the frequencies f_1 to f_5, as shown in the figure.

According to Miller, the main problem of using a gate array and departing from the difference equation model for digital filters is that we can not assume linearity anymore. We can quote him:

Unlike conventional filter design the (gate array) filters have not been designed using a linear difference equation and all the power of the mathematics of modern signal processing theory. Thus, we can assume nothing about the behaviour of the evolved filters.



(i)



(ii)

Figure 6.25 Frequency responses of the evolved low-pass gate arrays with the frequency response fitness (i) and with the error-based fitness (ii). (Extracted from Miller[13])

In order to draw more solid conclusions about the evolved filters behavior, he performed additional tests with the evolved gate arrays. In one of his tests, for instance, he presented to the evolved gate arrays an input consisting of the sum of two sine waves. In this experiment, the frequencies of the sine waves, f_1 and f_2, laid in the passing band, having equal amplitudes. Figure 6.26 shows the output and frequency response for both filters.

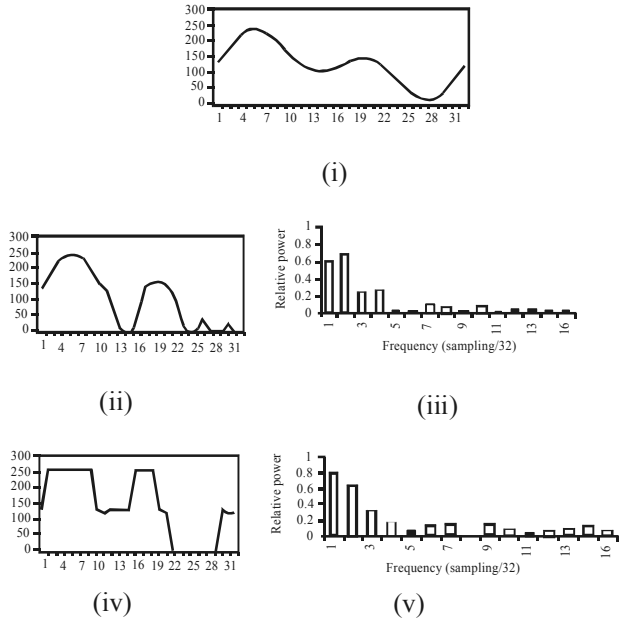


Figure 6.26 Incident signal 0.5 (f_1 + f_2): (i) output and frequency response for frequency based fitness gate array; (ii) output and frequency response for error-based fitness gate array.

REFERENCES

[1] Louis, S. J. and Rawlins, J. E., Designer genetic algorithms: genetic algorithms in structure design, ICGA-91, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, Belew, K.K. and Booker, L.B., Eds., Booker, Morgan Kaufman, San Manteo, CA, 1991, 53.

[2] Liu, W., Murakawa, M., and Higuchi, T., ATM cell schedule by function level evolvable hardware, in *Proceedings of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, vol. 1259, LNCS, Tsukuba, Japan, Springer-Verlag, October, 1997, 180.

[3] Zebulum, R. S., Pacheco, M. A., and Vellasco, M., Increasing length genotypes in evolutionary electronics, in http://www.ai.mit.edu/people/unamay/icgasubmissions.html, *ICGA'97 Workshop on Evolutionary Computation with Variable Size Representation*, July, 20, 1997.

[4] Hill, F. J. and Peterson, G. R., *Introduction to Switching Theory And Logical Design*, John Wiley & Sons, New York, 3rd ed., 1981.

[5] Aguirre, A.H., Coello, C.A.C., and Buckles, B.P., A genetic programming approach for logic function synthesis by means of multiplexers, in *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, IEEE Computer Society Press, Pasadena, CA, July 19-21, 1999, 46.

[6] Aguirre, A., Buckles, B. P., and Coello-Coello, C., Evolutionary synthesis of logic functions using mMultiplexers, *Proceedings of the 10th Conference on Smart Engineering Design (ANNIE 2000)*, St. Louis, November 2000, ASME press, pp. 311-316.

[7] Thompson, A., Silicon evolution, in *Proceedings of Genetic Programming 1996 (GP96),* Koza, J.R., et al., Eds., MIT Press, Cambridge, 1996, 444.

[8] Iba, H., Iwata, M., and Higuchi, T., Machine learning approach to gate-level evolvable hardware, in *Proceedings of the First Conference on Evolvable Systems: From Biology to Hardware*, vol. 1259, LNCS, 1997, Tsukuba, Japan, Springer-Verlag, October 1996, 327.

[9] Miller, J. F., Luchian, H., Bradbeer, P. V. G., and Barclay, P. J., Using a genetic algorithm for optimising fixed polarity Reed-Muller expansions of Boolean functions, *International Journal of Electronics*, 76, 1994, 601.

[10] Miller, J.F., Thomson, P., and Fogarty, T., Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: a case study, in *Genetic Algorithms Recent Advancements and Industrial Applications*, Quagliarella, D. et al., Eds., John Wiley & Sons, New York, 1997.

[11] Koza, J.R., *Genetic Programming*, MIT Press, Cambridge, 1992.

[12] Schaffer, J. D., and Eshelman, L. J., Designing multiplierless digital filters using genetic algorithms, *Proceedings of the Fifth International Conference on Genetic Algorithms*, ICGA-93, Forrest, S., Ed., Morgan Kaufmann, San Manteo, CA, 1993, 439-444.

[13] Miller, J., On the filtering properties of evolved gate arrays, in *Proc. of The First NASA/DoD Workshop on Evolvable Hardware*, Stoica, A., Keymeulen, D., and Lohn, J., Eds., Pasadena, 1999, 225.

[14] Schaffer, J. D. and Eshelman, L. J., Designing multiplierless digital filters using genetic algorithms, in *Proceedings of the Fifth International Conference on Genetic Algorithms*, ICGA-93, Forrest, S., Ed., Morgan Kaufmann, San Manteo, CA, 1993, 439.

[15] Brayton, R. K., Hachtel, G. D., Hemanchandra, L., Newton, R., and Sangiovanni-Vincentelli, A., A comparison of logic minimisation strategies using ESPRESSO: an APL program package for partitioned logic minimisation, in *Proceedings of the International Symposium in Circuits and Systems*, Rome, Italy, April 1982, 42.

[16] Ellis, M., *Electronic Filter Analysis and Synthesis*, Artech House, 1994.

CHAPTER 7

Evolution of Circuits on Reconfigurable Chips

This chapter introduces the field of reconfigurable chips and modern applications of these devices. We present two classes of reconfigurable devices, Field Programmable Gate Arrays (FPGAs) and Field Programmable Analog Arrays (FPAAs). Modern applications of these devices involve the evolutionary design of electronic circuits and Virtual Computing. Recently GAs have been employed as an agent that programs a reconfigurable chip in order to carry out automatic circuit design. Promising results have been attained, which contributed to increase the interest in this area of research, commonly called Evolvable Hardware (EHW). On the other hand, the idea of Virtual Computing is based on the use of highly reconfigurable and adaptive computer systems, realized through FPGAs. The main challenge in this area is merging Evolvable Hardware and Virtual Computing.

Reconfigurable chips are integrated circuits whose internal connections can be programmed by the user. Field Programmable Gate Arrays (FPGAs) and Field Programmable Analog Arrays (FPAAs) constitute the state of the art in the technology of reconfigurable chips, referring to digital and analog devices, respectively. These devices will be the building blocks of a forthcoming class of hardware which will present two important properties: the self-adaptation and self-repairing features, through automatic reconfiguration. These properties are important for systems that have to work for a long time in harsh environments. For instance, the aerospace community would be one to benefit from this new class of hardware systems. The automatic reconfiguration of these devices, which promotes the self-adapting and self-repairing features, may potentially be driven by Evolutionary Computation techniques.

This chapter is organized in four sections. The main subject of the first section is FPGAs. We provide a description of Programmable Logic Devices (PLDs), emphasizing FPGAs, the most important reconfigurable logic devices at present. The second section presents the analog counterparts of the FPGAs, the FPAAs. FPAAs have just recently appeared, and most projects are being carried out in universities and research centers. The third section presents applications of Genetic Algorithms in the automatic design of analog and digital circuits, using reconfigurable chips as the evolving platform. This kind of application received the name of Evolvable Hardware (EHW). Finally, the last section of this chapter introduces a new area of research, Virtual Computing. Virtual Computing promises to endow ordinary computer systems with more flexibility due to the reconfigurability. This new property would be useful to adapt the computer hardware to the particular software it would be running. We also make a sketch of the future of the area, which will probably include the merging of Evolvable Hardware and Virtual Computing.

7.1 PROGRAMMABLE LOGIC DEVICES

A number of devices belong to the group of Programmable Logic Devices or PLDs. The more important examples of PLDs[1] are:

- PROM, *Programmable ROM*
- PAL, *Programmable Array Logic*
- PLA, *Programmable Logic Array*
- PLS, *Programmable Logic Sequencer*
- FPGA, *Field Programmable Gate Array*

This section briefly describes the first four devices of the above list with focus on the FPGAs. Before describing these devices, it is important to mention the original motivations for their appearance: facility for prototype development; low development cost; fast production; and facility to introduce design changes.[2] Also, a list of common terms employed in the context of PLDs is provided below:

- SPLD: simple PLD (PLA or PAL)
- CPLD: complex PLD (FPGA, superPAL, megaPAL)
- Logic Capacity: Number of two-inputs AND gates
- Logic Density: Amount of logic/unity of area
- Logic Block: regular structure observed in the FPGA
- Programmable switch: device that provides interconnectivity to the chip according to the user's programming

The last feature of the above list, the programmable switch, will characterize the PLDs in terms of its programmability. Table 7.1 surveys the main technologies utilized to implement the switches:

Table 7.1 Main programmable switches.[2]

Switch	Reprogrammable?	Volatility?	Technology
Fuse	No	No	Bipolar
EPROM	Yes	No	UVCMOS
EEPROM	Yes	No	EECMOS
SRAM	Yes	Yes	CMOS
Anti-fuse	No	No	CMOS+

According to the switch technology used to configure the PLD, the programmable device may be reprogrammable or not. In addition, an instantiated circuit may stay configured or not after turning off the power supply, according to the switch volatility property. We will now summarize the main features of the different PLDs.

7.1.1 PROM

PROMs were the first chips that could be programmed by the user. The inputs of this chip are the address lines, and the outputs are the data lines. PROM works as look-up tables, being able to implement any kind of combinational function. They are not volatile, but they can not be re-programmed. Figure 7.1 depicts the typical structure of a PROM, consisting of 16 words of 4 bits. As it can be observed, the PROM is composed by two plans: a fixed AND plan and a programmable OR plan.

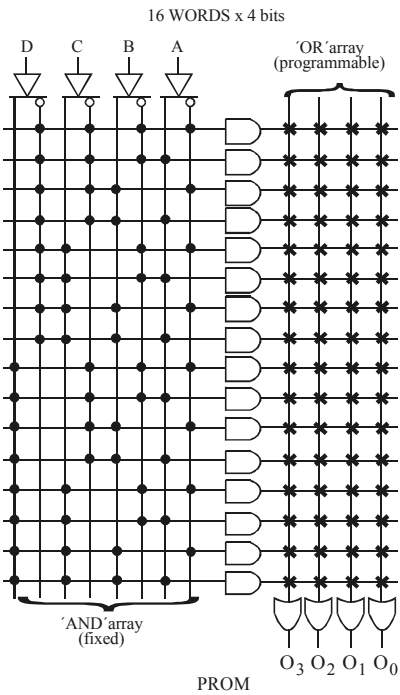
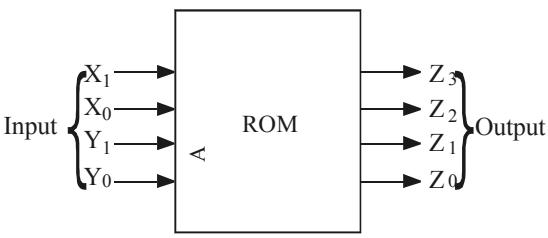


Figure 7.1 PROM structure. (Extracted from 3)

Figure 7.2 exemplifies an arbitrary combinational function, with 4 inputs and 4 outputs, being programmed in the PROM shown in Figure 7.1.

7.1.2 Programmable Logic Array (PLA)

The PLA structure is similar to the PROM's, in the sense that it consists of two plans, the AND and the OR plans. However, both plans are programmable in the PLA. This means that the user can configure the combination of input variables that will form each product term; and any sum of the realized product terms can be programmed in the OR plan. This endows PLAs with versatility in the implementation of logic functions. Figure 7.3 depicts the structure of a PLA.



Input	Outputs
$X_1\ X_0\ Y_1\ Y_0$	$Z_3\ Z_2\ Z_1\ Z_0$
0000	0000
0001	0000
0010	0000
0011	0000
0100	0000
0101	0001
0110	0010
0111	0011
1000	0000
1001	0010
1010	0100
1011	0110
1100	0000
1101	0011
1110	0110
1111	1001

Figure 7.2 Hypothetical example of a combinational function programmed in a PROM. (Extracted from 2)

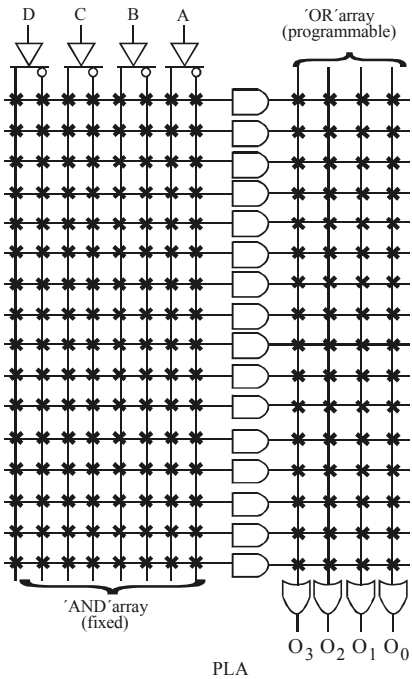


Figure 7.3 Structure of a PLA device with four inputs and four outputs.[2]

The main difference between the PLA and the PROM is that the former can not generate all the possible minterms, since the decoder is now replaced by the AND array. A PLA can be programmed either by the manufacturer (mask programming) or it can be programmed by the user (field programmable, or FPLAs).

The automatic design of a logic circuit in PLAs involves the following steps: simplification of Boolean functions' expressions, possibly through the use of software tools like Espresso; the selection of the smallest set of product terms that solve all the Boolean functions; and the PLA programming.

Previous to manufacturing, the PLA hardware can be sized according to the specifications of the circuit to be realized. The specification is given in terms of the product, *number of inputs x number of product terms x number of outputs*, or *n.k.m*. According to this representation, the PLA will be sized in the following way:

- *n* inverter buffers
- *k* AND gates
- *m* OR gates
- *m* XOR gates

- $2n \cdot k$ programmable connections in the input plan
- $k \cdot m$ programmable connections in the output plan
- m programmable connections associated to the XOR gates

In addition to the gates shown in Figure 7.3, the above list also includes XOR gates. XOR gates are usually connected to the PLA outputs in order to give the user the option to access the complemented value of the regular PLA outputs.

7.1.3 Programmable Array Logic (PAL)

The basic difference between PALs and PLAs is that only the AND plan is programmable, and the OR plan is now fixed. Therefore, they are not as versatile as PLAs. In order to make up for this lack of flexibility, PALs are usually manufactured in many different sizes. Figure 7.4 depicts the basic schematic of a PAL device.

By observing Figure 7.4, we can see that the outputs do not share product-terms, and there is a fixed number of product-terms per output (four in Figure 7.4).

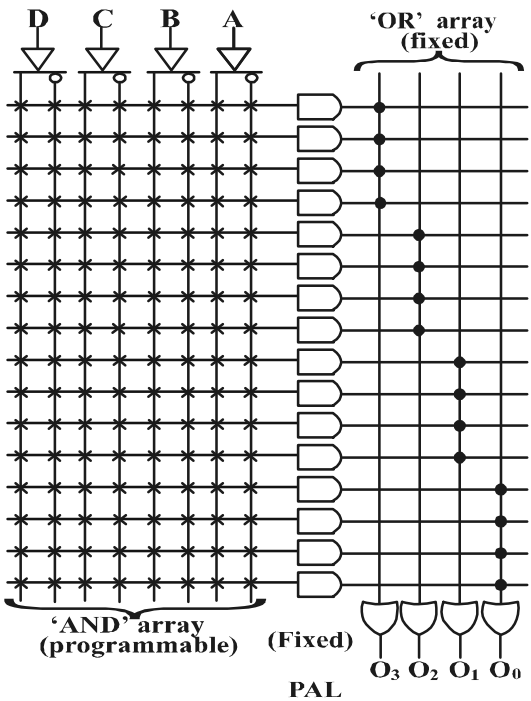


Figure 7.4 Structure of a PAL device with four inputs and four outputs. (Extracted from Reference 2)

By observing the above figure, we can see that the outputs do not share product-terms, and there is a fixed number of product-terms per output (four in the above figure).

PAL devices can also have flip-flops connected to their outputs, enabling the realization of sequential circuits, as shown in Figure 7.5.

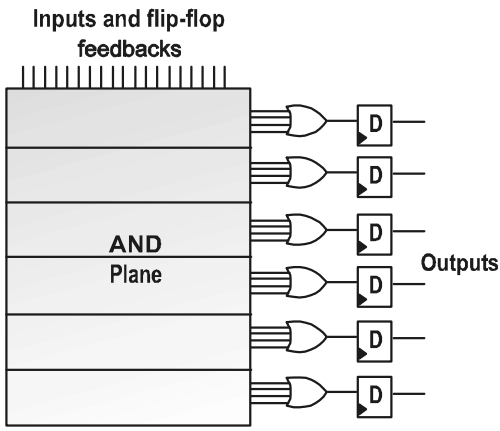


Figure 7.5 Structure of a PAL device with D flip-flops attached to its outputs. (Extracted from Reference 2)

7.1.4 Field Programmable Digital Arrays (FPGAs)

FPGAs are the most important programmable logic devices in the present moment: they are more advanced compared to the PLDs defined previously, and, from this perspective, they are more adequate for EHW applications than its predecessors. As described in Sanchez[3] (1996):

> *Field Programmable Gate Arrays (FPGA) are a recently developed family of programmable circuits. Like mask programmable gate arrays (MPGA) FPGAs implement thousands of logic gates. But, unlike MPGAs, a user can program an FPGA design as traditional programmable logic devices (PLDs): in-site and in a few seconds. These features, added to the reprogrammability, have made FPGAs the dream tool for evolvable hardware.*

FPGAs are VLSI chips with a high logic capacity, consisting basically of an array of logic blocks and interconnection resources that can be programmed by the user. They may utilize SRAM (*Xilinx* FPGAs) or anti-fuse (*Actel* FPGAs) switches, but in the latter they can not be reprogrammed. Due to the complexity of the logic blocks, software tools are required to program these devices. The more important FPGA manufacturers are *Xilinx*, *Actel*, *Altera,* and *QuickLogic*.

Figure 7.6 depicts the standard structure of an FPGA. Each rectangular box displayed in the figure represents a Configurable Logic Block (CLB). The lines surrounding the logic blocks represent the interconnecting resources, and the rectangular blocks in the device's boundaries are the I/O logic blocks. These logic blocks are specialized in routing input and output signals.

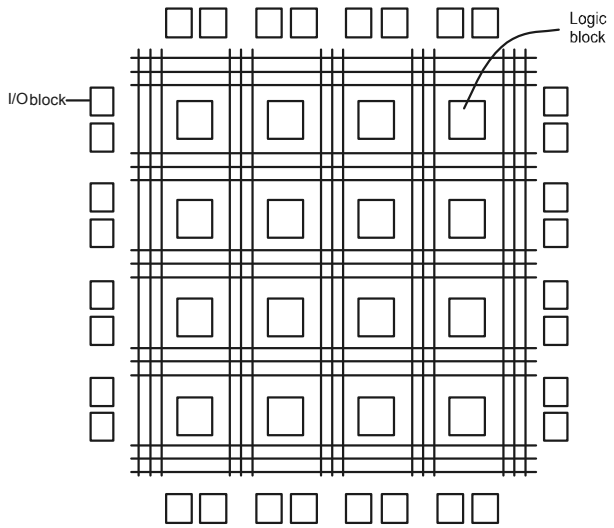


Figure 7.6 Basic structure of an FPGA. (Extracted from Reference 2)

Figure 7.6 depicts the schematic of a logic block from an FPGA manufactured by Xilinx. It can be observed that this logic block consists of:

- Look-up table: 2^k x1bit RAM to realize logic functions of k inputs
- Two flip-flops
- Logic gates
- Multiplexers and decoders to route signals
- Clock inputs

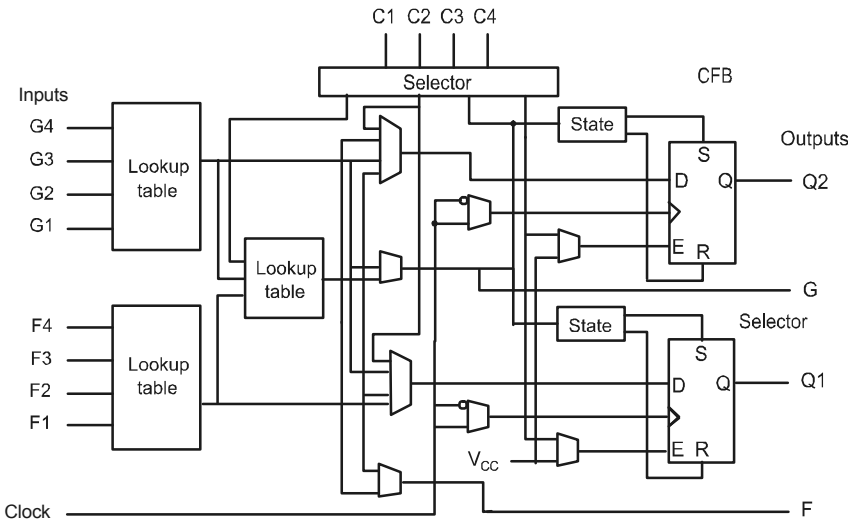


Figure 7.7 Configurable Logic Block (CLB) of an FPGA manufactured by Xilinx. (Extracted from[2])

Therefore, each logic cell can implement a wide variety of combinational and sequential logic functions. In addition to realizing logic functions, the logic blocks can be used only for routing purposes, through the multiplexers and decoders. As described in Sanchez[3] (1996), the interconnection structure of FPGAs can be of three kinds:

- Direct: the block outputs are directly connected to adjacent blocks.
- General Purpose: these are short lines placed between CLBs, horizontally and vertically. At an intersection point there is a switching matrix, allowing some interconnections between lines.
- Long lines: these lines cross all the circuit, horizontally and vertically, without using a switching matrix. Their number is smaller than that of general purpose lines and they are reserved for signals with critical timing.

In order to program an FPGA, the manufacturers provide special software tools in which the user only has to give the schematic or the netlist of the circuit. The tool will map the circuit into the FPGA resources, hiding from the user placement and routing details.

There has been a growing number of FPGA applications reported recently, including control tasks, communication coding applications, digital signal processing tasks, and innovative computing applications, such as Evolvable Hardware (EHW) and Virtual Computing (VC). The next sections of this chapter will focus on the latter two applications.

Finally, we conclude this section providing a brief description of the automatic design process of digital circuits based on FPGAs. Figure 7.8 depicts this process.

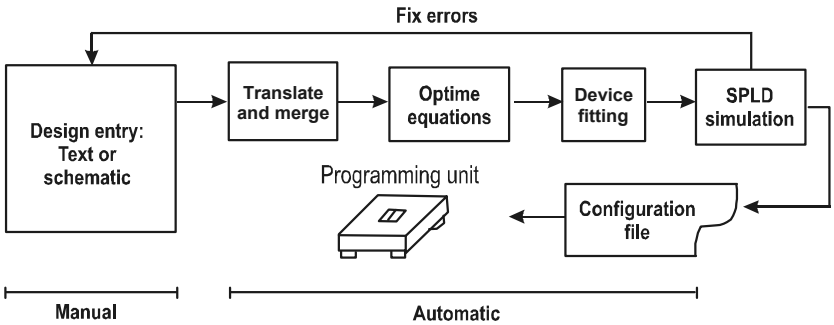


Figure 7.8 Block diagram of the framework utilized in the Computer Aided Design (CAD) of digital circuits based on FPGAs. (Extracted from[2])

We can enumerate some important features of the above figure:

- The design entry is given in the form of a schematic or in a hardware description language (HDL).
- There is a tool that carries out logic minimization, optimizing the design.
- The design is then adapted to the PLD (device fitting): this tool basically implements placement and routing, mapping the circuit into the logic blocks and interconnecting resources of the FPGAs.
- Before downloading the circuit, a simulation assures its correct operation.
- The circuit, represented by a bitstring, is then downloaded into the PLD.

7.2 FIELD PROGRAMMABLE ANALOG ARRAYS (FPAAs)

Contrary to PLDs, programmable analog circuits have only recently started to be investigated and fabricated.[4] It is interesting to note that many of these devices have been produced in university environments. This section will focus on the properties of these devices that enable them to be employed in Evolvable Hardware applications.

FPAAs can be classified into coarse and fine granularity devices. The former is composed by an array of high level analog components, such as operational amplifiers; the latter is constituted of an array of basic analog devices, typically transistors. This issue is discussed in Stoica[5] et al. (1998):

> *The optimal choice of elementary block type and granularity is task dependent. At least for experimental work in evolvable*

> *hardware, it appears a good choice to build reconfigurable hardware based on elements of the lowest level of granularity. Virtual higher-level building blocks can be considered by imposing programming constraints. An example of this would entail forcing groups of elementary cells to act as a whole (e.g. certain parts of their configuration bitstrings with the interconnections for the N transistors implementing a NAND would be frozen). Ideally, virtual blocks for evolution should be automatically defined/clustered during evolution (an equivalent of the Automatically Defined Functions predicted and observed in software evolution).*

We will describe seven analog reconfigurable devices in this section, four of them being coarse grained (Zetex, Motorola, Palmo, and Lattice) and the other three being fine grained (Evolvable Motherboard, PTA, and the Analog Multiplexer Array).

7.2.1 Totally Reconfigurable Analog Hardware (TRAC)

This device, manufactured by Zetex, is a coarse grained FPAA, consisting of twenty operational amplifiers laid out as two parallel sets of ten interconnectable amplifiers.[6] Each amplifier has various components around it that may be switched in and out of circuit by programming a shift-register to give one of the following distinct operations: off; pass; invert; exponentiate; take; logarithm; rectify; and sum of two inputs. Another characteristic of this programmable chip is the fact that resistors and capacitors must be externally connected by the user, since these components are not available on-chip. The Zetex programmable chip has been used in evolutionary experiments that targeted the synthesis of computational functions,[6] rectifiers, and others.[7] The main limitation of the use of this chip in EHW is that evolution is constrained to the arrangement of high level conventional building blocks to compose the final topology, not being able to arrive at any novel design.

7.2.2 Motorola

In 1997, Motorola[8] introduced the chip MPAA020, a coarse grained FPAA. The analog resources of this chip are distributed along 20 cells almost identically. Each cell is denominated Analog Configurable Block. Each block includes the following hardware resources:

- One CMOS operational amplifier
- One comparator
- Capacitors

- Programmable switches
- SRAM

This is the hardware required for the design of switched capacitor based circuits. Appendix B describes the fundamentals of Switched Capacitors Technology. This FPAA is configured by a bitstring of 6864 bits that control the circuit connectivity, capacitors' values, and other features. A single block can be used to implement amplifiers, adders, subtractors, rectifiers, sample and hold circuits, and first-order filters. More complex functions, such as biquad filters, PLLs, and level detectors, can be implemented using two or more cells from this chip.

Figure 7.9 depicts the basic scheme for connecting the Motorola FPAA to a PC. The software EasyAnalog (for Windows) provides a schematic editor, where the user can design a particular switched-capacitor circuit and then load it into the reconfigurable chip. The configuration takes place through the serial port that transmits 6864 bits to the FPAA board. Once the circuit is loaded (around 3 seconds time), its output can be visualized through an oscilloscope with probes directly attached to the programmable chip board. Signal generators are also attached to specific board pins, providing the input signal to the configured circuit. The circuit output is acquired by the PC through commercial analog boards for data acquisition. This board is usually connected to the PC bus, and its hardware includes sample and hold circuits and analog-to-digital converters.

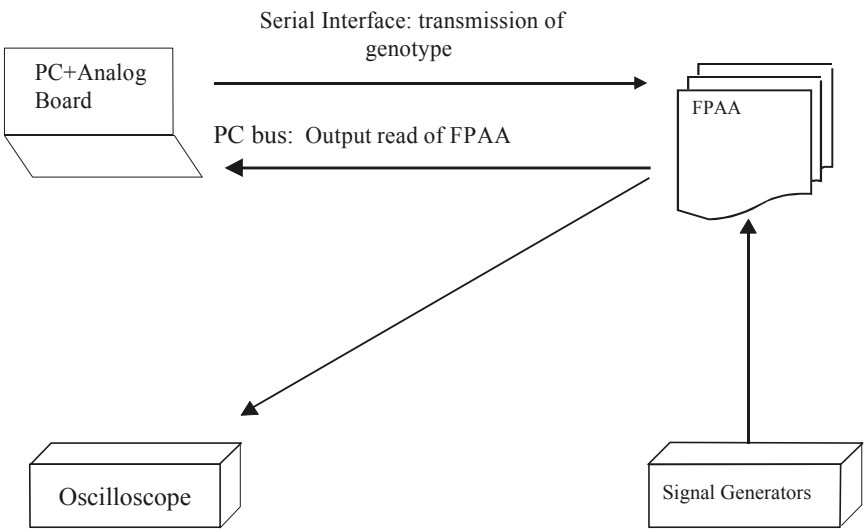


Figure 7.9 Framework for the use of the Motorola FPAA.

Another important concern is the one of illegal connections that could potentially destroy the reconfigurable chip, for instance, by shortening ground with the power supply. These dangerous configurations are filtered out previously to actually being downloaded, by the EasyAnalog[9] software tool.

One drawback of using the MPAA020 in EHW is that the basic building blocks, Operational Amplifiers (OpAmp), leave little room for evolution to arrive at novel designs. Another limitation of the current version of this chip is that it is not a transparent architecture, preventing simulation/analysis of evolved circuits.

7.2.3 Palmo (University of Edinburgh)

Another coarse-grained FPAA chip is the one devised by the research group of the University of Edinburgh, the Palmo development system. This chip works on pulse-based techniques, which constitute a distinct approach to implement signal processing functions.

Instead of representing signals as voltages or currents, digital pulses are used to represent discrete analog signals in this programmable analog array. Figure 7.10 illustrates an example of this kind of signal representation.

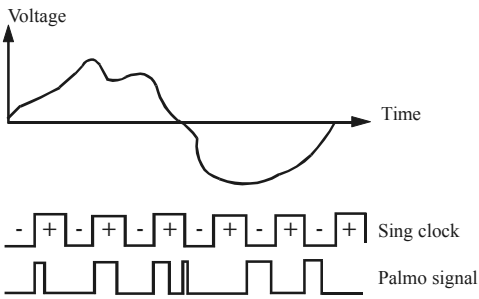


Figure 7.10 Hypothetical case of pulse-based signal representation in the Palmo system. (Extracted from Reference 10)

The representation shown above is explained in Hamilton[10] et al. (1998) in the following way:

"...the magnitude of the (discrete analog) signal is encoded in the width of the pulse. The sign of the signal is determined by whether the pulse occurs in the positive or negative cycle of a global sign clock. ... Due to the significance of the signal representation we have named these systems Palmo systems, where Palmo is Hellenic for pulse beat, pulse palpitation or series of pulses."

The Palmo chip is constituted by an array of programmable cells that perform the functionality of integrators. The integrator gain is programmed by a digital signal consisting of nine bits. Currently, the Palmo development system consists of a board containing Palmo chips, a Xilinx FPGA, and a microcontroller, as described by the designers:[16]

> *The pulsed signals for each Palmo cell and all the digital control signals are fed straight off chip and into a standard Xilinx FPGA. The FPGA may be programmed with digital logic circuits that control the functionality of the Palmo cells and that control the interconnection of the Palmo cells. Thus Palmo*

> *cells may be connected together and programmed to perform various analog signal processing functions. ... This development system contains two Palmo chips each containing 8 programmable analogue Palmo cells, a Xilinx FPGA, a microcontroller and some analog electronics for power supply management and reference signal generation for the Palmo chips. The host PC downloads the programming bitstream for the FPGA via the microcontroller. This sets up the control circuits and interconnections for the Palmo array. The host PC then downloads the K factors for the Palmo cells and the circuit is ready to operate.*

The *K* factors mentioned above refer to the integrators' gain. As reported in Hamilton,[10] 1998, recent Palmo devices have been implemented in BiCMOS technology, being operated at 5 Volts and at a sampling frequency of 5MHz. As the designers remark, the use of BiCMOS technology will possibly allow the system to be operated at sampling frequencies around 20MHz.

The mixed signal nature of the Palmo systems allows the use of digital circuits to manipulate the pulses representing analog quantities. An example given by the system's designers is the use of a digital circuit to multiply an incoming signal by a constant (a0 equal to 5 in Figure 7.11 where this example is illustrated).

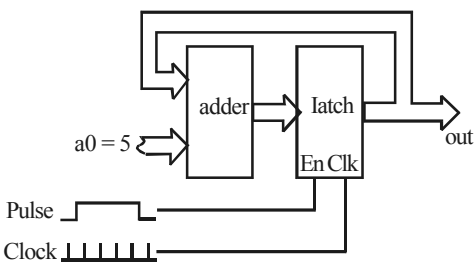


Figure 7.11 Palmo signal multiplied by a constant, a0 = 5. (Extracted from Reference 10.)

In the circuit of the above figure, the latch is initially cleared. While the pulse signal is high, the constant 5 is added to the contents of the latch output at every clock cycle. As the width of the pulse is proportional to its magnitude, at the end of the incoming pulse the latch output will consist of a digital word that represents the result of the multiplication of the pulsed signal by the constant.

Conversely, a digital word may be converted to a Palmo (pulse) signal , and fed into the programmable analog cells. This can be accomplished by using a down counter: if this counter is parallel loaded with the digital word to be converted, the time the counter takes to count down to zero can be used to generate the pulse.

One of the design goals is to use the chip in EHW experiments, employing GAs to determine the integrators' interconnectivities and their gains.

7.2.4 Evolvable Motherboard (University of Sussex)

The Evolvable Motherboard (EM) is a research tool developed at the University of Sussex, intended for the implementation of artificial evolutionary experiments as described in (Thompson et a.l, 1999):[11]

> *The tool (henceforth referred to as an Evolvable Motherboard (EM)) allows a large variety of electronic components to be used as the basic active elements, and has an interconnection architecture such that any component pin can be independently connected to any other. Interconnections are directly accessible by test equipment, facilitating analysis of circuits configured on EM.*

Figure 7.12 shows a simplified schematic of the Evolvable Motherboard.

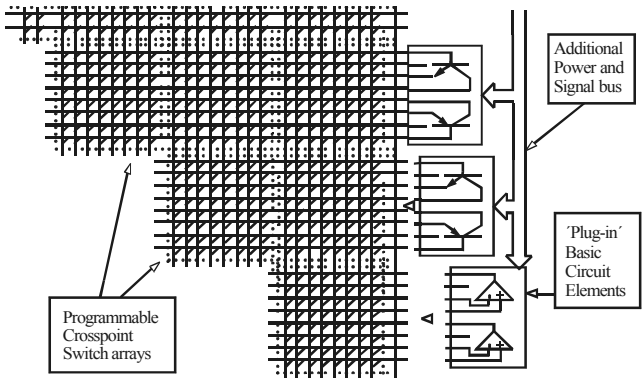


Figure 7.12 Simplified schematic of part of the EM (extracted from Thompson, Layzell, Zebulum, Fig. 29).

The above figure is a simplified diagram of one corner of the EM and plug-in daughter-boards containing the basic elements. The diagonal lines represent digitally controlled analog switches which allow row/column interconnection. The number *s* of switches required to ensure all possible combinations of interconnections is:

$$s = \frac{n(n-1)}{2}$$

where *n* is equal to the total number of basic element pins.

The EM explores non-idealities of the analog switches, such as parasitic resistances and capacitances. This is described in Thompson[11] et a.l, 1999:

> *Connections made using the analog switches have resistance and capacitance, hence forming an integral part of any circuit configured. In total, approximately 1500 switches are used,*

> *giving a search space of 10^{420} possible circuits. The 'on' resistance of the analog switches prevents configurations which short the power rails from damaging the EM, provided the power supply is less than 3VDC.*

The EM uses an ISA interface to download the genotype, so that the switches can be programmed by direct writes to a PC's internal I/O ports, allowing circuits to be instantiated in hardware in a short time, around 1ms. In order to access the output of the instantiated circuit, a commercial analog board is used. This analog board provides sampling and analog-to-digital conversion.

The daughter-boards are plugged into the motherboard and they have two analog components attached to them. These analog components will then be connected through the programmable crosspoint switch array. The user can choose the type of components to attach to the daughter-board: transistors, multiplexers, operational amplifiers, and others. So far, experiments have been performed using fine granularity elements, mainly bipolar transistors.

7.2.5 Programmable Transistor Array (PTA) (Jet Propulsion Lab)

The Programmable Transistor array is a fine grained FPAA developed at the Jet Propulsion Lab, being specifically targeted for experiments in Evolvable Hardware. The basic analog devices are MOS transistors, now integrated in a chip using 0.5-micron CMOS technology. This FPAA is described by Stoica[12] et al. (1999):

> *The PTA is a concept design for hardware reconfigurable at transistor level. As both analog and digital CMOS circuits ultimately rely on functions implemented with transistors, the PTA appears as a versatile platform for the synthesis of both analog and digital (and mixed-signal) circuits. Further, it is considered a more suitable platform for synthesis of analog circuitry than existing FPGAs or FPAAs, extending the work on evolving simulated circuits to evolving analog circuits directly on the chip. The PTA module is an array of transistors interconnected by programmable switches. The status of the switches (ON or OFF) determines a circuit topology and consequently a specific response. Thus, the topology can be considered as a function of switch states, and can be represented by a binary sequence, such as "1 0 1 1 ...", where by convention one can assign 1 to a switch turned ON and 0 to a switch turned OFF.*

Figure 7.13 illustrates an example of a PTA module consisting of 8 transistors and 24 programmable switches. This module consists of PMOS and NMOS transistors,

and switch based connections. The number of switches allows the configuration of many different topologies, but does not cover all possible transistor arrangements.

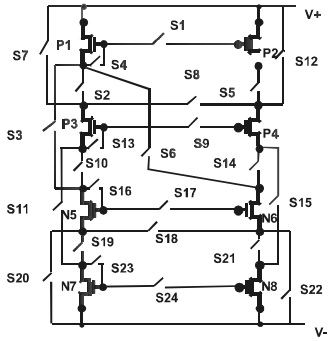


Figure 7.13 Module of the Programmable Transistor Array (extracted from Stoica[5] (1998b)).

The programmable switches were implemented with transistors, acting as transmission gates. This kind of switch can pass analog signals, and its resistance varies from tens of Ohms to hundreds of MOhms. The PTA designers remark that, for a first approximation, the switches' non-idealities can be neglected.

The PTA chip is built around LabView National Instruments data acquisition hardware and software.[13]

As explained above, although we classify the PTA in the domain of the Analog Arrays, this device can actually be used for the evolution of both analog and digital functions, since both of them ultimately rely on transistors.

One of the more important features of the PTA chip is the speed it confers to evolutionary experiments. The PTA designers compared the speed of an evolutionary experiment implemented in two frameworks: directly on the PTA chip (intrinsic approach) and using an HP supercomputer through a circuit simulator (extrinsic approach). In the former, each individual (circuit) of the GA is downloaded and evaluated in hardware. This approach will be explained in detail in the next section. In the second framework evolution is carried on using the SPICE simulator, as in the applications described in the last chapter. In the second framework, each individual is evaluated in parallel using the resources of the HP parallel supercomputer. The particular experiment lasted five seconds in the PTA chip and 20 minutes in the supercomputer. The chip designers comment on this issue:

> *Such models require evaluation with circuit simulators such as SPICE; the simulators need to solve differential equations and, for anything beyond simple circuits, they require too much time for practical search of millions of circuit solutions. A hardware implementation offers a big advantage in evaluation time for a circuit; the time for evaluation is determined by the goal function. For example, considering an A/D converter operating at a 100 kHz sampling rate the electronic response of the A/D converter is available within 10 microseconds compared to an (over-optimistic) 1 second on a fast computer running*

SPICE; this advantage increases with the complexity of the circuits. In this case the 10^5 speedup would allow evaluations of populations of millions of individuals in seconds instead of days.

Another interesting feature of the PTA is the possibility of mapping circuits of higher complexity, while still being programmable at the transistor level. It has been shown in Zebulum[14] et al. (2000) that standard circuits for analog and digital systems, such as tranconductance amplifiers and logic gates, can be implemented using the PTA architecture depicted in Figure 7.13. This provides the designer with the possibility to choose the most adequate building blocks to be manipulated by the evolutionary algorithm.

7.2.6 Programmable Analog Multiplexer Array (PAMA) (Catholic University of Rio de Janeiro)

The Programmable Analog Multiplexer Array (PAMA) is essentially a fine granularity FPAA, even though its building blocks can be chosen by the user, from transistors, resistors, and capacitors, to higher level circuits, such as operational amplifiers and comparators.

The design of this programmable analog platform has been inspired by previous work,[15,16] in which the designers have used a Genetic Algorithm (GA), based on integer representation, in which each gene encodes a particular component: its nature, value, and connecting points.

The analog reconfigurable circuit is divided into three layers: discrete components, analog multiplexers, and analog bus. These layers are represented below in Figure 7.14.

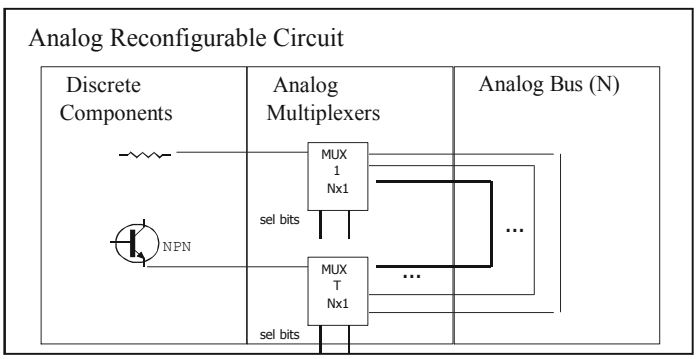


Figure 7.14 Analog reconfigurable circuit's layers.

Each component terminal is plugged to a certain line of an analog bus through an analog multiplexer. The chromosome bits, which are sent by the PC, select for each analog multiplexer a certain line of the analog bus to connect to the component's

terminal. Each line of the analog bus corresponds to one interconnection point of the circuit; some of them can be associated with external points, such as the input signal, power supply, ground, and circuit output, while others are associated with internal points of interconnection of the circuit. For each discrete component terminal there has to be an analog multiplexer *Nx1*, where N is the number of lines in the analog bus or connecting points.

The idea of a programmable Multiplexer Array was introduced first in Zebulum, et al.[17] (2000), where a platform's prototype implementing a 8-channel analog multiplexer/demultiplexer (bi-directional) SN74LV4051A was presented. The original platform communicates with a PC through its serial port to receive configuring bits and to transmit A/D converted signals, at a transmission baud rate of 19200 bps.

As this platform concept proved to be adequate to intrinsically evolve analog circuits, another platform has been designed to overcome two main drawbacks of the prototype: the data acquisition bottleneck and small number of discrete components.

This new PAMA platform consists of a bigger analog reconfigurable circuit with a faster interface. The analog reconfigurable circuit has a 32 16-to-1 analog multiplexer/demultiplexer (bi-directional). Its low on-resistance (Ron) of 100Ω prevents the analog reconfigurable circuit from damage during random configurations. As shown in Figure 7.15, a multifunction I/O board connected to the PC bus is responsible for the A/D conversion and for the chromosome download.

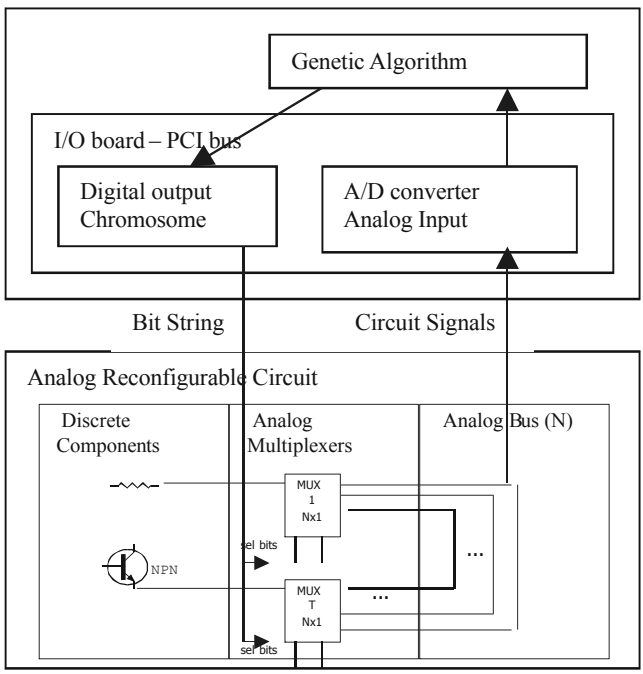


Figure 7.15 The new PAMA platform.

PAMA's correspondent search space is P^T, where T is the number of component terminals (32) and P is the number of circuit connecting points or analog channels (16). Then $P^T = 16^{32} \sim 10^{38}$. Some PAMA features are presented below:

- Programmable granularity: PAMA allows all discrete components to be selected by the user, according to the application
- Transparent architecture: circuits which have evolved on PAMA can be easily analyzed, once on-board signal acquisition can be made
- Evolved circuits can be easily reproduced, just following the interconnection points and including the multiplexers on-resistance
- Protection against damaging configurations: PAMA's parasitic resistances of the switches (Ron) provide this protection
- In PAMA, evolution is able to exploit any interconnection pattern or architecture, thus allowing exploitation of richer topologies, dynamical behaviors, and complex aspects of semiconductor physics[18]
- The number of the evolved circuit's input and output signals is not fixed and can be determined according to the application
- This platform design can be easily scalable. For instance, to evolve a circuit having 10 transistors, 10 resistors, and seven capacitors, the platform would need 64 multiplexers, resulting in a chromosome length of 320 and the correspond search space of 32^{64}. Instead, this can be done with four platforms, in which each gene will consist of 5 bits ($2^5 = 32$)

7.2.7 Lattice PAC

The Lattice PAC is a coarse-grained programmable analog circuit that was commercially introduced recently,[6] intended for applications in filtering, summing/differencing, gain/attenuation, and conversion. The PAC cell includes instrumentation amplifiers, an active resistance, and a programmable capacitor. The major limitation for the application of this chip to EHW is the fact that it is programmed through non-volatile memory cells with limited number of program/erase cycles, around 100,000, insufficient for evolutionary experiments.

7.2.8 Comparative Table

We provide here a comparative table with the main features of the analog reconfigurable circuits presented above.

Table 7.2 Comparison among the main features of the FPAA devices reported in the literature.

	TRAC	MPAA020	PALMO	EM	PTA	Lattice	PAMA
Granularity	Coarse	Coarse	Coarse	Fine	Fine	Coarse	Fine
Protection	----------	Software tools	Soft/hard	Switches	Hardware architecture	----------	Switches
Download	Parallel port	serial port	Port	ISA bus	Bus	Serial port	Parallel port
Probing	External analog acquisition board	External analog acquisition board	----------	External analog acquisition board	National Instruments	External Analog Acquisition Board	On-board analog acquisition
Search Space	----------	~2^{300}/cell	----------	10^{420}	2^{24}/cell	----------	10^{38}
Technology	CMOS	CMOS	BiCMOS	Board Level	CMOS	CMOS	Board Level

7.3 EVOLVABLE HARDWARE

The use of Genetic Algorithms as a tool for automatic synthesis of circuits on reconfigurable platforms received the name of *Evolvable Hardware (EHW)*. The reader may also find, in the literature, the terminology Evolvable Hardware used in a broader sense than the one employed in this book: some authors use EHW to refer to any general application. We prefer to restrict the term Evolvable Hardware to the applications involving evolution on reconfigurable chips. Another commonly used terminology for Evolvable Hardware is *Intrinsic Evolution.*[19] This contrasts with the Extrinsic Evolution, performed through the use of circuits' simulators.

We describe in this section digital and analog EHW applications, respectively corresponding to the use of FPGAs and FPAAs. We start presenting the pioneering experiment of Thompson,[20] which boosted the field of Evolvable Hardware. He evolved a tone discriminator using a FPGA, achieving a circuit far more economic than human designed ones. As one conclusion of his experiments, Thompson pointed out that a promising development of EHW would involve the use of FPAAs for intrinsic evolution. This fact encouraged many researchers to investigate the issue of intrinsic evolution on FPAAs. This section describes several experiments related to evolution on the FPAA devices introduced previously in this chapter.

7.3.1 Thompson

We describe here the first successful attempt to evolve a circuit on a FPGA platform through Genetic Algorithms. This experiment has been reported in Thompson's DPhil thesis, concluded at the University of Sussex, in England.[20] His experiments consisted of using a GA to evolve a frequency discriminator on a Xilinx manufactured FPGA. Following, he describes his experiments:[11]

> *The task was to evolve a circuit – a configuration of a 10x10 corner of the XC6216 FPGA – to discriminate between square waves of 1kHz and 10kHz presented at the input. Ideally, the output should go to +5V as soon as one of the frequencies is present, and 0V to the other one. The task was intended as a*

> *first step into the domains of pattern recognition and signal processing, as well as being part of a scientific programme. One could imagine such a circuit being used to demodulate frequency-modulated binary data received over a telephone line.*

As it is remarked by Thompson, this task would be quite easy for a human designer once he has access to external resources, such as clocks or RC time constants. But, imagine if an experienced human designer is given the task of making this project using only basic digital gates, such as AND, or, and XOR. This task would be nearly impossible with the use of conventional tools to digital circuit design. The difficulty of this task is better explained in Thompson[11] et al. (1999):

> *Evolution was required to produce a configuration of the 100 cells to discriminate between input periods five orders of magnitude longer than the input, using the output propagation time of each cell (which is just a few nanoseconds). No clock, and no off-chip components could be used: a continuous time recurrent arrangement of the 100 cells had to be found which could perform the task entirely on-chip.*

In the following, we describe the experiment setup, results and analysis.

7.3.1.1 Experiment Setup

A genotype length of 1800 bits was used to configure the portion of the gate array used in the experiment. This resulted in a huge search space of 2^{1800} possible circuits (around 10^{540}). The GA used a population of 50 individuals, a crossover probability of 0.7, and a per-bit mutation probability such that the expected number of mutations per genotype was 2.7.

The execution of this experiment involved an interface of a PC with the FPGA. This is shown in Figure 7.16.

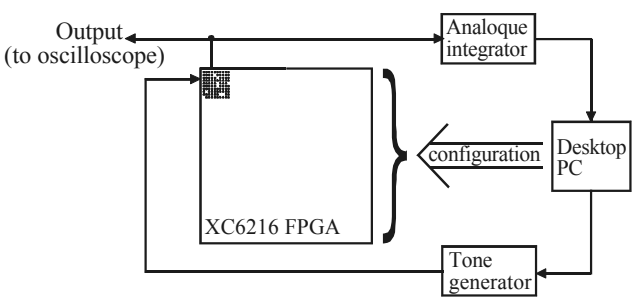


Figure 7.16 Apparatus for the tone discriminator experiment. (Adrian, Layzell, Zebulum, Fig. 15.)

The circuit was mounted on an ISA card that could be directly plugged into the PC. The bitstring is downloaded into the FPGA board from the PC, as indicated in the above figure. A PC-controlled tone generator drove the circuit's input with five 500ms bursts of the 1kHz square-wave and five of the 10kHz wave. The ten tones were presented to the circuit in a random order that changed along the generations. The analog integrator was used to integrate the voltage at the circuit's output pin over the 500ms duration of the tone.

The fitness evaluation function stimulated the maximization of the difference between the average output voltage when a 1kHz input is applied and the average output voltage when a 10kHz input is applied. He used the following equation:

$$fitness = \frac{1}{5}\left| \left(k_1 \sum_{t \in S_1} i_t \right) - \left(k_2 \sum_{t \in S_{10}} i_t \right) \right|$$

In the above equation i_t represents the integrator reading at the end test tone t. S_1 is the set of 1kHz tones and S_{10} is the set of 10kHz tones. k_1 and k_2 were empirically determined constants.[11]

7.3.1.2 Results

Figure 7.17 displays the photographs, taken throughout the experiment, of an oscilloscope screen attached to the circuit output. From this figure the improving behavior of the best individual of the population along the generations can be observed. At generation 3500, an individual is found that can accomplish the task, with an output of 0 Volts for the 1kHz tone, and 5 Volts for the 10 kHz tone. However, though not visible in the figure below, the circuit at generation 3500 is not perfect, due to some unwanted spikes at the output. Proceeding with the evolutionary process, a circuit was found at generation 5000, whose output accomplished the discrimination task without any spikes or delays when the input changed from one tone to another.

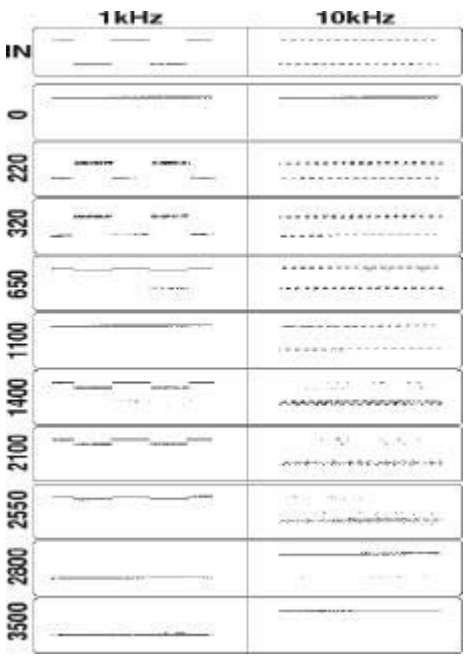


Figure 7.17 Photographs of the oscilloscope screen along the tone discriminator experiment. The 1kHz and 10kHz waveforms are displayed at the top; the corresponding output of the best individual in the population is displayed below; the generation number is written at the left of the graphs. (Extracted from Thompson, Layzell, Zebulum, Fig. 17.)

This experiment took 2-3 weeks. This amount of time was basically due to the individual's evaluation time, around five seconds per individual. The final circuit consisted of a 10x10 array of cells. Nevertheless, not all of these cells were effectively contributing to the circuit's behavior. Figure 7.18 shows the functional part of the circuit, where only the cells that affect the circuit's output appear.

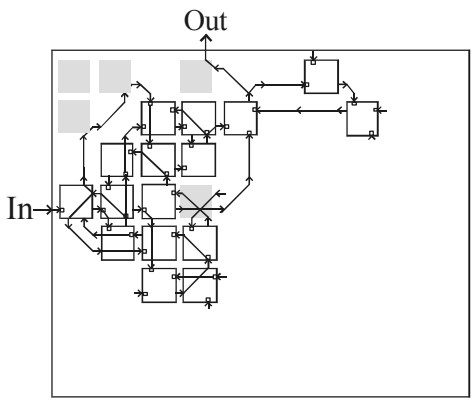


Figure 7.18 Functional part of the circuit evolved in the tone-discrimination experiment. (Extracted from Thompson, Layzell, and Zebulum, Fig. 21.)

As it can be seen, only 21 cells are actually contributing to the circuit's behavior. Five of these cells are shaded gray in this figure, since they produce a notable feature. Although there is no path connecting them to the output, they influence the circuit's behavior. According to Thompson, the gray cells are probably interacting with the circuit by some other means than the normal inter-cell routing; this issue is discussed in detail in Thompson and Layzell,[21] 1999.

This has been an impressive result, which stimulated other scientists to investigate the field of Evolvable Hardware. Evolution was able to find a very small circuit to perform a task that would require human designers to project larger and clocked circuits, or use bulky components, such as capacitors and inductors. This is described in Thompson[11] et al. (1999):

> *This circuit is discriminating between inputs of period 1ms and 0.1ms using only 32 cells, each with a propagation delay of less than 5ns, and with no off-chip components whatsoever: a surprising feat. Evolution has been free to explore a full repertoire of behaviours available from the silicon resources provided, even being able to exploit the subtle interactions between adjacent components that are not directly connected. The input/output behaviour of the circuit is a digital one, because that is what maximising the fitness function required, but the complex analogue waveforms seen at the output during the intermediate stages of evolution betray the rich continuous-time, continuous-value dynamics that are likely to be present.*

We can summarize the statement above by saying that this outstanding result is explained by two facts: the analog behavior of the FPGA; and the use of feedback. As remarked by Thompson, although the FPGA's logic gates should, in principle, work as amplifiers in either saturation or cut-off regions, they are actually working in the linear region in this experiment, allowing the exploration of analog properties. If conventional digital design techniques are not used, and this is certainly the case of evolutionary design, then one can not assure that the logic gates will work in the digital mode. In fact, evolution used a digital programmable device in the analog mode to perform the proposed task. In addition, even though clocks and flip-flops were not employed in this experiment, feedback between logic gates was allowed. This type of feedback certainly generates a rich repertoire of behaviors that can not be easily simulated, since they are dependent on the propagation delay of each cell. It is very difficult for a human designer to explore these delays in a project, because they can not be determined with sufficient precision, varying for different gates of the FPGA as well.

7.3.1.3 Analysis

The next step followed by Thompson was the analysis of the evolved circuit. The schematic of the circuit is drawn in Figure 7.19. This schematic is constituted by logic gates, and it is based on the assumption that all of the FPGA's cells are acting as Boolean logic gates in the normal way. However, as the experiment's results suggest, this assumption may be not a correct one.

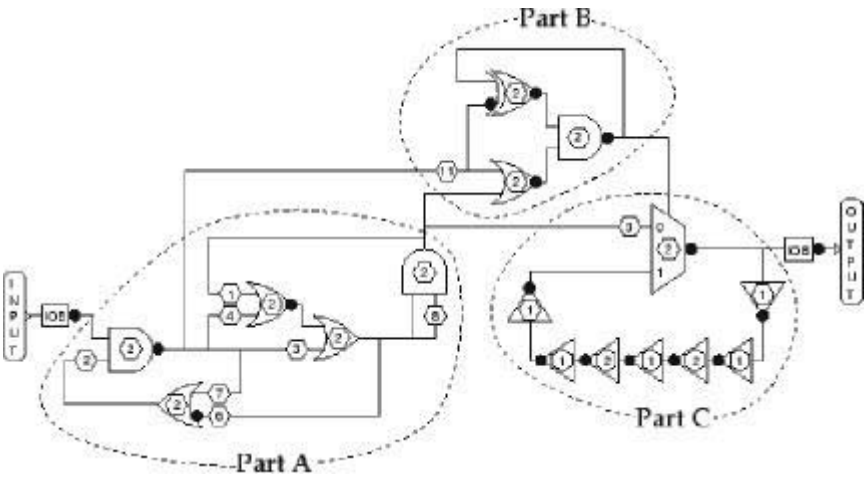


Figure 7.19 Representation of the evolved tone discriminator through logic gates. The number in hexagons are estimates of the time delays (in ns). IOB refers to the inverting Input/Output blocks that drive the input and output signals (Thompson, Layzell, Zebulum, Fig. 23).

The above circuit is analyzed by Thompson[11] et al. (1999):

> *The logic circuit diagram shows several continuous-time recurrent loops (breaking the digital design rules), so the system's behaviour is unlikely to be fully captured by this Boolean level of abstraction. However, it contains many 'canalysing' functions, such as AND and OR: functions where one input can assume a value which makes the other inputs irrelevant. It so happens that whenever our circuit's input is 1, all of the recurrent loops in Parts A & B are broken by a cascade of canalysing functions. Within 20ns of the input becoming 1, A and B satisfy the digital design rules, and all of their gates deterministically switch to fixed, static logic values, staying constant until the input next go to 0.*

After performing an extensive study with the help of the PSPICE simulator, the circuit's behavior and the function of the above mentioned "gray cells" could be partially understood,[21] as described by Thompson:

> *We performed a digital simulation of the circuit (using PSPICE), extensively exploring variations in time-delays and parasitic capacitances. The simulated circuit never reproduced the FPGA configuration's successful behaviour, but did corroborate that the transient as the circuit enters its static state at the beginning of an input is just a few of nanoseconds, in agreement with our experimental measurements of internal FPGA signals, and according with logic analysis. We then built the circuit out of separate CMOS multiplexer chips, mimicking the way that the gates are actually implemented by multiplexers on the FPGA, and also modelling the relative time-delays. Again, this circuit did not work successfully, and – despite our best efforts – never produced any internal activity during an input pulse.*
>
> *We understand well how parts B & C use A's initial oscillatory dynamics to derive an orderly output, and have successfully modeled this in simulation. The time delays on the connections from A to B & C are crucial. This explains the influence of the 'gray cells', which are immediately adjacent to (or part of) the path of these signals. Varying the time delays in simulation produces a similar result to interfering with the gray cells. Mostly, the loop of part C serves to maintain a constant and steady output even while the rest of the circuit oscillates, but immediately after an input pulse it has subtle transient dynamics interacting with those of A & B.*

Another important remark of Thompson referred to the robustness of this circuit. It has been observed that the circuit's performance degraded with changes in temperature in a larger extent than conventional circuits would; and when this circuit was downloaded into another (identical) FPGA chip, it also exhibited a different performance.

> *From the observations made so far, one could only speculate as to the circuit's means of operation, so unusual are its structure and dynamics. It was clear that continuous time played an important role. If the circuit was configured onto a different, but nominally identical, FPGA chip (not used during evolution), then its performance was degraded. If the temperature is raised or lowered, then the circuit still works, but the frequencies between which it discriminates are raised or lowered. (Digital circuits usually display unchanged behaviour followed by brittle failure on approaching their thermal limits.)*

The explanation for this fact is that evolution has deeply explored the physical properties of the silicon to achieve the solution. One drawback of this fact is the circuit's lack of robustness. In order to overcome this problem, Thompson built an evolutionary framework where the circuit's robustness would also be considered. Instead of downloading each individual in only one FPGA, he downloaded it in five different FPGAs. These FPGAs are manufactured from different foundries, and they are also exposed to different temperature conditions. The fitness evaluation combines the five scores achieved corresponding to the circuit behavior in the different FPGAs.

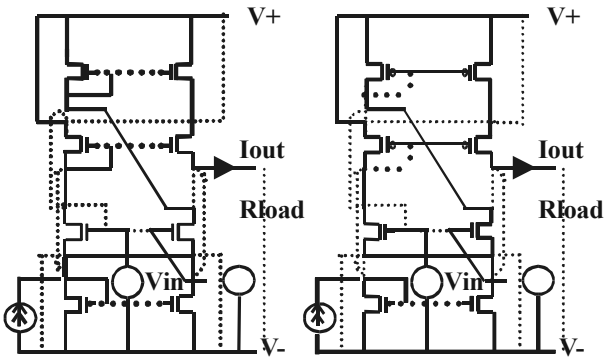
7.3.2 Stoica et al.

The Programmable Transistor Array under test at the Jet Propulsion Laboratory has been applied to the evolution of a Gaussian computational function. This application is described in Stoica[12] et al., 1999:

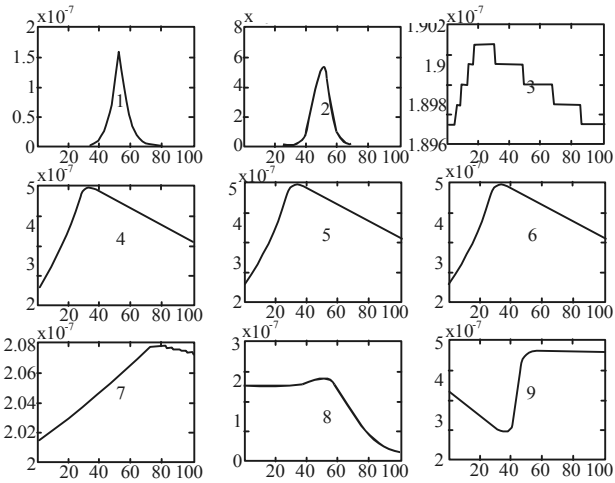
> *The goal of evolution was to synthesize a circuit which exhibits a Gaussian I-V (current-voltage) input-output characteristic. In a previous experiment the circuit topology was fixed and the search/optimization addressed transistor parameters (channel length and width); such evolution proved quite simple. In the FPTA case, the transistor parameters were kept fixed and the search was performed for the 24 binary parameters characterizing switches status. Successful evolution was achieved in multiple runs with populations between 50 and 512, evolving for 50 or 100 generations. The execution time depends on the above variables and on the number of processors used (usually 64 out of the 256 available), averaging around 20 minutes (the same evolutions took about 2 days on a SUN SPARC 10).*

As described in the above paragraph, the first attempt to the evolution of this Gaussian circuit was accomplished through extrinsic experiments, using a simulated model of the PTA. In these experiments, evolution was able to rediscover a conventional circuit that exhibited this kind of transfer function. In addition, variants of the conventional circuit were also found by the Genetic Algorithm.

Figure 7.20 depicts two circuits evolved in the cited experiment. It also shows some of the transfer functions of the best circuits achieved by the GA.



(A)



(B)

Figure 7.20 (A) Circuits obtained by evolution; their design is unusual for common practice; (B) best circuit responses in a simulated evolution. (Extracted from Stoica[12] et al. (1999).)

About the results shown in the above figure, the PTA designers commented:

It is interesting to analyze in more detail the unusual solutions found by evolution. Circuits like those illustrated in Figure 20.A resulted from evolutionary synthesis, and are very similar (under certain test conditions) to the human designed solution. Thicker dotted lines show connections that existed in the circuit

> *in the human designed circuit, but are missing in the circuits in Figure 20.A. Figure 20.B shows the DC transfer of some circuits that resulted from this experiment: the first two responses correspond to the circuits in (A). Some other responses from the same generation are illustrated for comparison. As it is easy to observe, these circuits are outside normal design practices, e.g., the transistors P2, P4 and N8 on the left circuit in Figure 9.A have floating gates. The reality is that the switches have a big, but finite, resistance in the Off state (~MOhms or GOhms) and a non-zero resistance/impedance in the On state (~ tens of Ohms). One observation from here is that while the effects of non-perfect switches may be negligible in a first approximation for many digital circuits, such effects may fundamentally affect analog programmable circuits.*

This experiment was repeated in hardware. The results obtained through intrinsic evolution were akin to the ones achieved by extrinsic evolution. The following setup was used in the intrinsic experiment:[12]

> *Four chips were programmed in parallel with bit-string configurations corresponding to four individuals of a population of 100; then, the next four were programmed and so on, until all 100 in one generation were tested. As in simulation, evolution led to "Gaussian" circuit solutions within 200-300 generations in 4.5 min using the four PTA chips in parallel.*

7.3.3 Zebulum et al. (2000)

The PAMA platform was tested in two phases: first using the PAMA prototype, then using the new PAMA platform.

In the first phase, a reduced Analog Reconfigurable Circuit, with a minimum number of discrete components, was used to evolve the circuit of a inverter: three discrete components, nine 8-to-1 analog multiplexers, and 8-line analog bus.[14]

The Genetic Algorithm used in this test presents a traditional setting:

- Binary Representation: 27 bits
- Population Initialization: Random
- Reproduction Technique: Steady State
- Operators: One-point Crossover and Mutation

- Fitness Technique: Exponential Normalization

The fitness function to evolve the inverter preemies circuits with maximum difference between input and output: Fitness = Σabs[Vin(i)–Vout(i)].

In a first test, the genetic algorithm parameters were: population size of 200; 10 generations (representing 0.0015% of the search space of $P^T = 8^9$); crossover rate of 0.65; and the mutation rate of 0.10. The discrete components were three NPN transistors and a 4K7W resistor which was placed in series with the power supply in order to help achieve the DC level. The circuit in Figure 7.21 corresponds to the best-evolved solution, which was obtained in the 10th generation after running the genetic algorithm for approximately 1 hour in a Pentium II 233MHz.

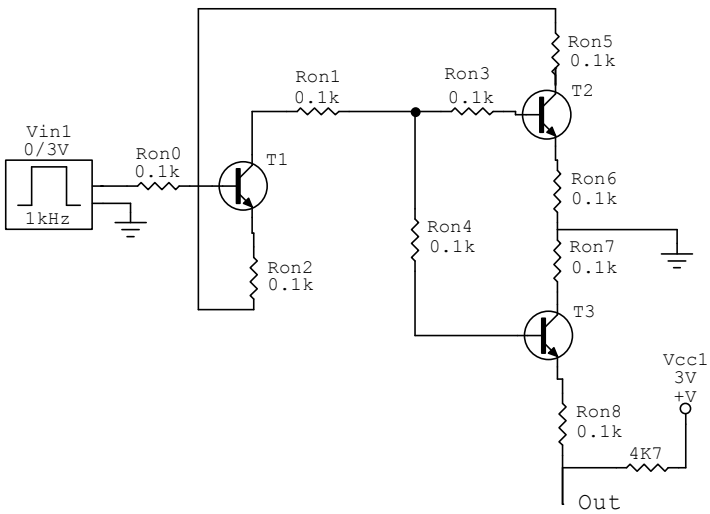


Figure 7.21 Inverter circuit evolved.

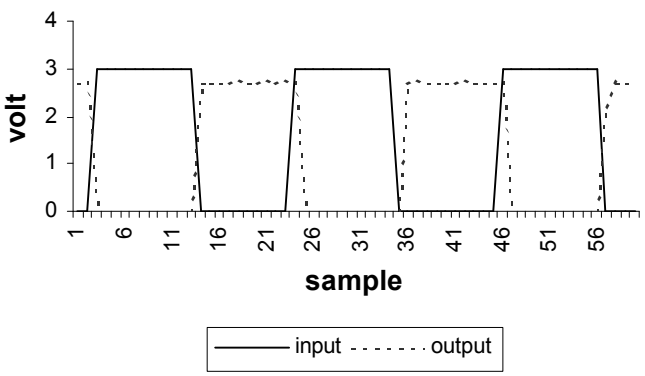


Figure 7.22 Input and output signals of the inverter.

Figure 7.22 contrasts the waveforms of the input and the output signals of the evolved inverter in Figure 7.21. The input signal is a 1 KHz, 0 to 3V square wave. Input and output signals were alternately transformed by two distinct A/D converters, before being used by the genetic algorithm. The process of reading, converting, and writing back to the microcontroller's memory takes approximately 25 μs, and is repeated 20 times to each period of these signals. The transmission baud rate between the FPAA and the PC is 19200 BPS. As the microcontroller deals with one A/D converter at a time, there exists a 25 μs delay between the samples of input and output signals, which affects evaluation in points near to the signals' transitions.

In another test, the genetic algorithm parameters were: population size of 50; 40 generations (representing 0.012% of the search space of 8^8); crossover rate of 0.65; and the mutation rate of 0.10. Here we used the 4K7Ω resistor as a discrete component instead of placing it in series with the power supply. The other two discrete components were two NPN transistors. The chromosome representation in this case required 24 bits.

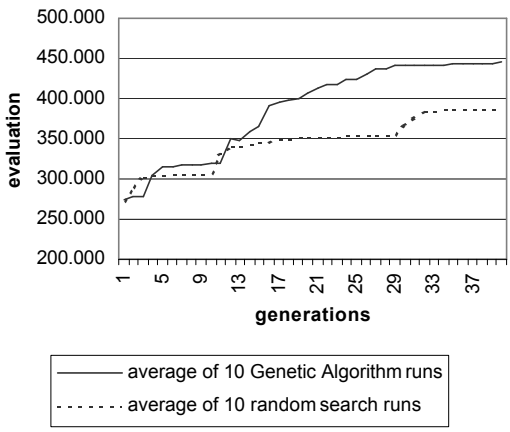


Figure 7.23 GA evolution against Random Search.

Figure 7.23 shows the evolutionary system performance through the average of the best circuit evaluation in 10 executions, against the average of the best circuit evaluation in 10 random search executions.

The best circuits evolved in the 1st generation had the average evaluation of 273.412 and its output signal was meaningless regarding an inverter. In the 20th generation the best circuits evolved presented an average evaluation of 407.473 and performed as an inverter. However, the output V_{pp} was less than 2V.

The circuit in Figure 7.24 corresponds to the evolved solution which presented the closest evaluation to the average evaluation (446.543) of the 40th generation.

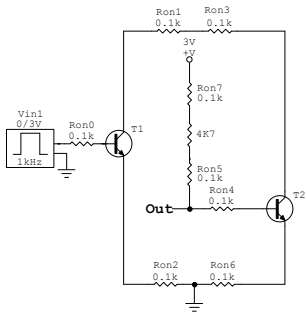


Figure 7.24 Inverter circuit evolved on the second test.

Figure 7.25 contrasts the waveforms of the input and the output signals of the evolved inverter in Figure 7.24.

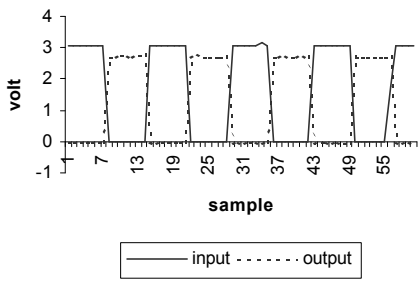


Figure 7.25 Input and output signals of the inverter on Figure 7.24.

In the second phase, the new PAMA, a larger Analog Reconfigurable Circuit (32 16-to-1 analog multiplexer/demultiplexer), described in section 7.2.6, was used to evolve a XOR gate.[22] The faster interface (I/O board) was not used in this preliminary test. Due to that, it took considerable time (one day, two runs) to obtain results.

The Genetic Algorithm employed in this XOR evolution presented a similar setting, except for the chromosome representation, now with 128 bits (32 x 4).

In this experiment, 8 transistors and 4 resistors were chosen. The fitness was calculated using Mean Square Error. The error was the difference between the desired output and the real output. The desired output was computed according to the XOR logic. Figure 7.26 shows the two input signals and the evolved circuit's output signal versus the desired signal. Figure 7.27 shows the evolutionary system performance through the average of 10 genetic algorithm runs.

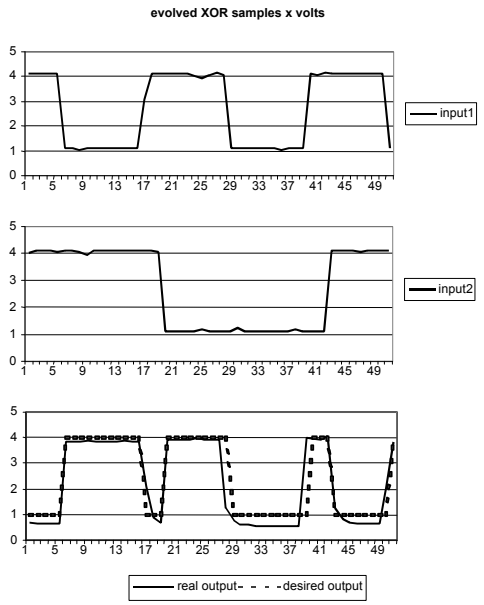


Figure 7.26 Best XOR evolved input and output signals.

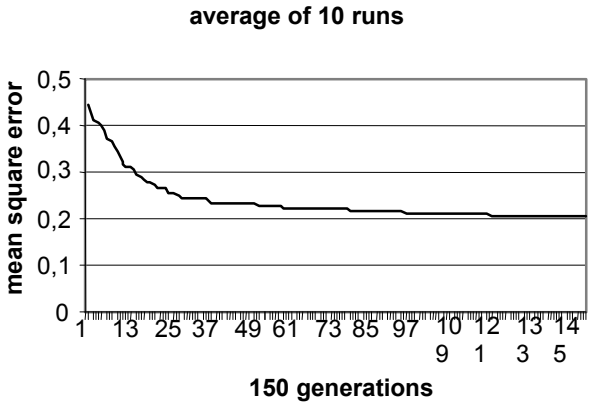


Figure 7.27 Mean Square Error average in 10 GA runs.

7.3.4 Zebulum et al. (1999)

The FPAA device from Motorola, MPAA020, was used by the authors in the implementation of EHW experiments. We recall that this is a coarse-grained reconfigurable device based on switched capacitors technology. This chip consists of 20 cells, each of which contains operational amplifiers, programmable capacitors,

and switches. Many evolutionary experiments have been performed using this device, encompassing the synthesis of oscillators, filters, rectifiers, and amplifiers. We will describe in this section an experiment targeting the evolution of a half-wave rectifier.

A half-wave rectifier can be conventionally implemented using only one cell of this device.

Thus, we used only one device cell in the experiment. The genotype size necessary to configure one cell was 870 bits, producing a huge search space of 10^{261} possible circuits. The GA had the following parameters: population size of 200 individuals; single-point crossover rate of 70%; and mutation rate of 9% per bit. The fitness evaluation function is given by the following equation:

$$Fitness = 5000 - Max\left(\sum_{i=0}^{n-k} | s_o(i+k) - s_t(i) |\right), k = 0,1,2,\ldots,\frac{f_{samp}}{f_{in}} - 1$$

In the above equation s_o represents the actual output samples and s_t represents the target output vector. In this particular case s_t consists of a half-cycle rectified sine wave. The difference between the actual and the target output samples is computed along the total number of acquired samples, n. Since this is an error measure, it is multiplied by –1. An arbitrary constant can be added for the sake of visualization only, 5000 in this case. A total of f_{samp}/f_{in} fitness measures are calculated for each individual, where f_{samp} is the sampling frequency and f_{in} is the input wave frequency. The final fitness score is the maximum value among these fitness measures. The variable k defines a phase shift between the actual and target output vectors, varying from zero to the number of samples associated to a period of the input wave. The objective of this procedure is just to align both waves in the time axis.

The experiment lasted around four hours in a 166 MHz PC. This time was basically dominated by the bitstring downloading time, around five seconds. Even when only a small part of the chip's analog resources is actually being configured, the whole binary string sequence must be sent through the serial connection, increasing the downloading time.

The experiment lasted around four hours. The graph of Figure 7.28 compares the best circuit output with the desired wave. A 1kHz sine wave was applied at the circuit's input, and the sampling frequency stayed at 40 kHz.

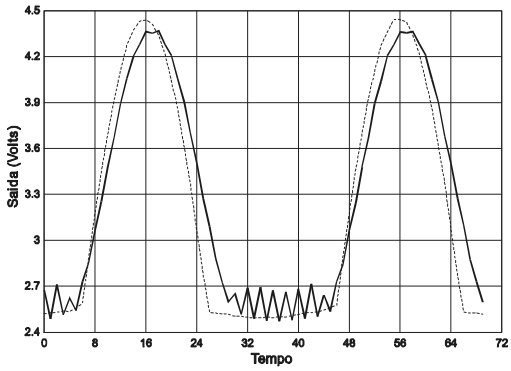


Figure 7.28 Comparison between the target (points) and evolved (continuum) waves in the half-wave rectifier experiment. The time unity (X axis) is 1/40 kHz.

From the above graph we can observe that the evolved circuit actually rectifies the sine wave. From this figure, we can also identify a noise added to the circuit's output. Observing in the oscilloscope, we found out that this noise was a low-amplitude 200kHz sine wave, not well reproduced in the figure due to its high frequency. Given the huge size of the search space associated with the experiment, we can say that this result was interesting.

One drawback of using the MPAA020 chip is impossibility the perform of an analysis of the evolved circuits, as this chip was not intended, in principle, to be used in EHW experiments. As a matter of fact, commercial FPGAs and FPAAs (mainly FPGAs) have been manufactured to be applied in a wide variety of applications, and EHW is a very peculiar one compared to the others. In order to use a reconfigurable device in an EHW experiment, one has to be able to directly download the bitstring into the device. However, the manufacturers provide software tools that convert the user schematic or HDL circuit representation into the binary string that program the device, hiding from the user all the details of the process. To perform EHW experiments it is necessary, therefore, to ask for special information from the device suppliers about direct access to the chip. More importantly, it is necessary to get information about invalid bitstrings that may damage the device by programming an illegal configuration in it. Besides direct chip access information, EHW experiments also require extra tools that enable the user to "read" the circuit instantiated by a particular bitstring. These tools must be supplied by the chip's manufacturers as well. In the case of the MPAA020, it was possible to directly download a hypothetical bitstring into the chip and filter out invalid configurations; but there were no tools to read circuits configured without the standard FPAA software package. This imposes a limit to the experiments, because it is not possible to make a complete analysis of the result.

7.4 VIRTUAL COMPUTING

Virtual computing is a paradigm for computation based on reconfigurable hardware, which contrasts to standard computing, which is based on fixed hardware systems. A formal definition of virtual or reconfigurable computing can be found in VCC,[23] 2000:

> *Reconfigurable computing systems are those computing platforms whose architecture can be modified by the software to suit an application at hand. To obtain maximum throughput, an algorithm must be placed in hardware. Dramatic performance gains are obtained through 'hardwiring' of the algorithm. In a reconfigurable computing system, the 'hardwiring' takes place on a function by function basis as the application executes.*

The main idea expressed in the above definition is: instead of having your general purpose hardware system running different software applications, let the hardware adapt according to the particularities of the software being executed. Naturally, in order to have such a flexibility, programmable digital circuits are required at the lower level of the system. We will then have a hardware that is tuned to the particular algorithm, nearly as if it was manufactured to implement that particular algorithm. This tuning is accomplished by programming the reconfigurable logic in a particular way to optimize each application execution.

Currently, reconfigurable computers use plug-in boards that work as reconfigurable co-processors. These co-processors will execute computationally intensive tasks more efficiently than the main CPU. Figure 7.29 presents a block diagram of a reconfigurable computer.

The reconfigurable processor unit (RPU) acts as a co-processor to the main processor unit. The RPU is a programmable hardware, configured to execute a particular piece of code in an optimized way. The programmability of the RPUs derives from the fact that they are based on SRAM FPGAs.

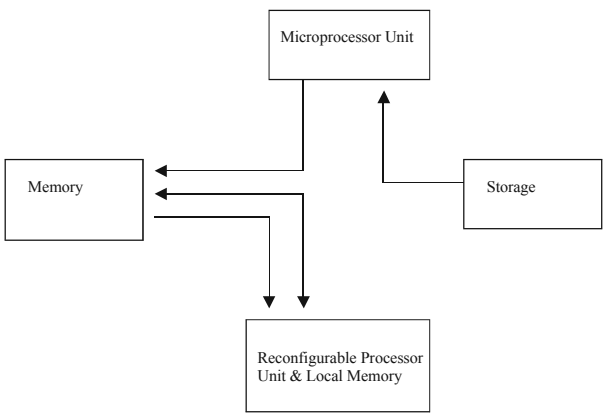


Figure 7.29 Block diagram of a reconfigurable computer (Extracted from (VCC,[23] 2000).)

The field of Virtual Computing would get many benefits from the automatic hardware reconfiguration feature promoted by EHW. The reconfigurable processor unit would, in this case, evolve to meet the requirements of a particular algorithm. Instead of having a library of fixed hardware objects, we would now have a library of evolved hardware objects. The potential advantages of this evolved library would be:

- The computing system would now be self-programmable: from a set of high level instructions,[12] artificial evolution would find a hardware algorithm that accomplishes the instructions.

- The new hardware object would have the potential of using less hardware resources than the conventional counterparts. For instance, as it was depicted in Chapter 6, the evolved hardware implementation of digital filters is based on simple digital gates, instead of adders and shift registers. Likewise, many algorithms would now be implemented by fine grained digital hardware resources manipulated by evolution.

- Evolution can also synthesize the control logic of the RPU in such a way to improve the performance of particular algorithms. The evolutionary design of the control logic of a CPU is one of the topics of the next chapter.

REFERENCES

[1] Mano, M.M. and Kime, C. R., *Logic and Computer Design Fundamentals*, Prentice-Hall, Eaglewood Cliffs, NJ, 1997.

[2] Brown, S. and Rose, J., FPGA and CPLD architectures: a tutorial, *IEEE Design & Test of Computers*, 1996, 44.

[3] Sanchez, E., Filed programmable gate array (FPGA) circuits, in *Towards Evolvable Hardware: The evolutionary engineering approach*, Sanchez, E. and Tomassini, M., vol. 1062, Eds., Springer-Verlag, LNCS, 1996, 1.

[4] Gaudet, V. and Gilak, G.., CMOS Implementation of a current conveyor-based field programmable analog array, in 31st ASILOMAR Conference on Signals and Systems and Computers, Pacific Grove, CA, November 1997.

[5] Stoica, A., Fukunaga, A., Hayworth, K., and Salazar-Lazaro, C., Evolvable hardware for space applications, in *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, Lausanne, Switzerland, September, 1998, 23. Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 166.

[6] Flockton, S. and Sheehan, K., Intrinsic circuit evolution using programmable analog arrays, in *Proc. of the Sec. International Conference on Evolvable Systems*, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 144.

[7] Ozsvald, I., Short-circuiting the design process: evolutionary algorithms for circuit design using reconfigurable analog hardware, Msc. Thesis, School of Cognitive and Computer Sciences (COGS), University of Sussex, 1998.

[8] Motorola Semiconductor Technical Data. Advance Information Field Programmable Analog Array 20-Cell Version MPAA020. Motorola, Inc.

[9] Easy Analog Design Software User's Manual, Motorola Inc., 1997.

[10] Hamilton, A., Papathanasiou, K., Tamplin, M. R., and Brandtner, T., Palmo, Field programmable analog and mixed-signal VLSI for evolvable hardware, in *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, Lausanne, Switzerland.

[11] Thompson, A. and Layzell, P., Analysis of unconventional evolved electronics, to be published in a special session of Communications of the ACM, Xin Yao, Ed., April 1999.

[12] Stoica, A., Keymeulen, D., Tawel, R., Salazar-Lazaro, C., and Li, W., Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital CMOS circuits, in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, IEEE Computer Press, July 1999, pp.76.

[13] LabView Evaluation Package, Graphical Programming and Instrumentation, National Instruments, Austin, Texas.

[14] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., A novel multi-objective optimisation methodology applied to synthesis of CMOS operational amplifiers, *Journal of Solid-State Devices and Circuits*, Microelectronics Society - SBMICRO, Noije, W. V. , Ed., February 2000, vol. 8, 1.

[15] Zebulum, R.S., Pacheco, M.A., Vellasco, M., A multi-objective optimisation methodology applied to the synthesis of low-power operational amplifiers, in *Proceedings of the XIII International Conference in Microelectronics and Packaging*, vol. 1, Chueiri, I. J. and Filho, C. A., Eds., Curitiba, Brazil, August 1998, 264.

[16] Zebulum, R.S., Pacheco, M.A., and Vellasco, M., Evolutionary systems applied to the synthesis of a CPU controller, to be published in *Lectures Notes in Artificial Intelligence*, Springer-Verlag, 1999, Second Asia-Pacific Conference on Simulated Evolution and Learning (SEAL98), Canberra, Australia, November 1998.

[17] Zebulum, R.S., et al., A reconfigurable platform for the automatic synthesis of analog circuits, in *The Second NASA/DoD Workshop on Evolvable Hardware*, IEEE Computer Society, Los Alamitos, Silicon Valley, California, July 13th, 2000, 91.

[18] Layzell, P., A new research tool for intrinsic hardware evolution, in *Proceedings of the Second International Conference on Evolvable Systems,* Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., LNCS, Springer-Verlag, 1998. Lattice Semiconductor Corporation, ispPAC Handbook, Programmable Analog Circuits, Sep., 1999, vol. 1478, 47.

[19] DeGaris, H., Evolvable hardware: genetic programming of a Darwin machine, *Artificial Neural Nets and Genetic Algorithms*, Albretch, R.F., Reeves, C.R., and Steele, N.C., Eds., Springer-Verlag, New York, 1993.

[20] Thompson, A., Harvey, I., and Husbands, P., Unconstrained evolution and hard consequences, in *Towards Evolvable Hardware: An International Workshop*, Lausanne, Swiss, 1995, 2. Chapter of book: Towards Evolvable Hardware: The evolutionary engineering approach, Sanchez, E. and Tomassini, M., Eds., Springer-Verlag LNCS 1062, 1996, 136.

[21] Thompson, A., Layzell, P., and Zebulum, R., Explorations in design space: unconventional electronics design through artificial evolution, *IEEE Trans. on Evolutionary Computation*, vol. 3, 3, 1999, 167.

[22] Santini, C.C., et al., *Evolution of Analog Circuits on a Programmable Analog Multiplexer Array*, IEEE Aerospace Conference, Big Sky, Montana, March 2001, 10.

[23] Virtual Computer Corporation. The Reconfigurable Computer Company. What are reconfigurable computers?, in http://www.vcc.com/intro2.html.

CHAPTER 8

Advanced Topics of : Filtering, Control, Antennas, and Fault Tolerance

This chapter describes some special topics of , selected by the fact that they are closer to real world applications. This set of applications covers the synthesis of active filters for mobile phones, the evolution of control circuits using different levels of granularity, evolution of antennas, and the evolutionary synthesis of fault tolerant circuits.

So far this book described how evolutionary computation could be applied to synthesize a wide variety of analog and digital circuits. However, the reader may ask the following questions: When will this technique deliver industrial applications? Would anyone have the courage to ride in an elevator or in a train with electronics control systems synthesized by artificial evolution? Naturally, it is very difficult to give an accurate answer to these questions; nonetheless, it is the belief of the EvolElec community that within some years, this will transform from an exciting area of research to a new technology. The applications selected in this chapter attempt to give a response to the first question, by describing the efforts of the researchers towards the goal of real world applications. The second question posed above is about how reliable evolvable systems can be. At the end of this chapter we show that, as a matter of fact, evolvable electronics have the potential to be more reliable than conventional electronics, due to its fault tolerance properties.

The case studies described in this section certainly demonstrate this. We show here how GAs can optimize active filters to be used in industry. In a second set of applications, the synthesis of control systems through Evolutionary Computation is described. In addition, we review an application where GAs are applied to the synthesis of an antenna that outperforms conventional designs. Finally, we discuss the issue of fault tolerant circuits and how evolution can be used in this kind of design.

8.1 ACTIVE FILTERS

We review here the application developed by Murakawa[1] et al. (1998) in the Electro-technical Laboratory (ETL) in Tsukuba, Japan. They have used Genetic Algorithms to optimize the transconductances of amplifiers in a band-pass filter employed as a receiver of mobile phones (IF filters). The objective is to compensate for variations in the manufacturing process.

This application contrasts with the ones described previously in this book that also focused on the evolution of filters. While the latter were a proof of concept, the former handles a problem encountered in the IC industry. Particularly, they tackle the optimization of a new category of filters based on G_m amplifiers.[2] The reader can refer to Appendix C for a description of this kind of continuous time filter. For now it is important to understand that they are based on transconductance amplifiers.

Due to manufacturing limitations, the transconductance varies greatly from the target values. According to Murakawa[1] et al. (1998), the variations due to temperature can be corrected by a Phase-Locked-Loop (PLL) circuit. However, other variations can not be corrected using this circuit, because of the relative differences between the tranconductance values.

The solution found by Murakawa[1] and fellows from ETL was to use GAs to further optimize the amplifier transconductances. This can be accomplished by changing the base currents of the G_m amplifiers. A reconfigurable chip was designed in such a way that the total value of the base current is controlled by switches. This is shown in Figure 8.1.

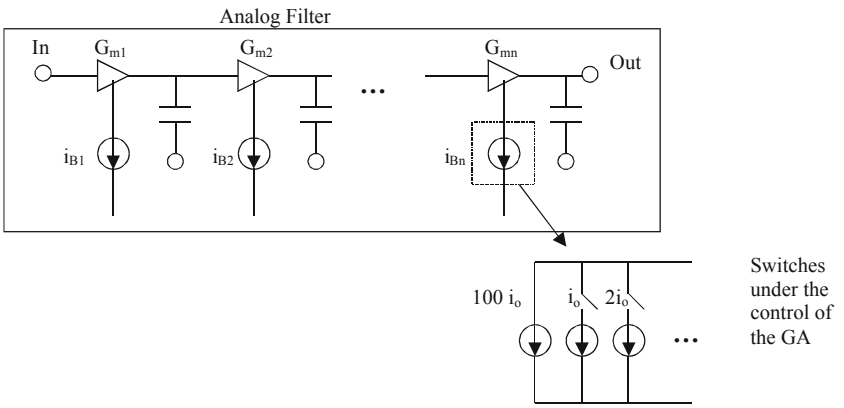


Figure 8.1 IF filter based on reconfigurable G_m amplifiers (extracted from Murakawa[1] et al. (1998)).

The IF filter has a band pass characteristic, with passing band between 440kHz and 470 kHz. The stop band is characterized by an attenuation around –60dB at the left and right sides of the frequency band (420 and 480 kHz). This roll-off is much steeper than the ones presented by the filters evolved in the previous chapters. Nevertheless, in this case, the GA already starts from a 18th order filter and the task is the optimization of the transconductances G_m amplifiers. Contrastingly, in the previous chapter, evolution synthesized both topology and component values of the filters, starting from scratch.

A GA with 50 individuals running 50 generations was employed to successfully improve the performance of the filter. The successful results of this application led to an industrial cooperation. This proposed chip will be manufactured and possibly used commercially in cell phones. That is an example of how technology is already starting to be commercialized.

Another meaningful result observed by Murakawa[1] and fellows was the superiority of GAs in comparison to the Hillclimbing method in this application. This result demonstrated the effectiveness of population-based search algorithms like GAs.

8.2 CONTROL APPLICATIONS

We describe in this section applications of Evolutionary Computation in the automatic synthesis of Proportional-Integral-Derivative (PID) controllers . The first application describes the use of Genetic Algorithms in the synthesis of controllers represented in the lowest level of granularity, the one of electric components. The second application describes the use of Genetic Programming to synthesize PID controllers represented through a higher level of granularity, the one of block diagrams. Interestingly, the SPICE simulator is able to simulate controllers in these two representation levels.

8.2.1 Evolution of Control Circuits Using Genetic Algorithms

GAs are utilized to synthesize circuits that perform control functions. This application also benefits from the energy minimization multi-objective optimization method, enabling the designer to consider specific circuit's implementation requirements, such as dissipation, intrinsic noise, and fault tolerance (shown at the end of this chapter), in addition to standard specifications for control systems, such as rise time and over-signal.

8.2.1.1 Representation

We use the same representation described previously in this book, where each chromosome is composed by a set of genes, each of which encodes for a particular component.

8.2.1.2 Evaluation

The energy minimization method has been applied. The circuits' specifications are divided into *general specifications for control systems* and *implementation requirements*. The former consists of the integral square error of the transient response, rise time, and over-signal. The latter consists of the circuit dissipation, intrinsic noise, and fault tolerance requirements. The circuits are evaluated using the transient and noise analysis features of the PSPICE simulator.

The synthesis of control systems is a mature area of research, and there exist conventional methods that have been successfully applied to the design and analysis of these systems, such as the root locus and state space.[1,2] The main drawback of these methods is the fact that they do not take into account hardware implementation requirements, such as circuit dissipation, noise, fault tolerance, self-repairing capabilities, and other aspects.

The conventional way to design a control circuit involves two steps: arriving at a Laplace transfer function for the compensator, through the use of the techniques mentioned above; and finding a circuit that implements the Laplace transfer function. Standard circuit topologies, based on operational amplifiers or on passive networks, are achieved through this method.

Our approach, instead, performs the control circuit synthesis in just one step, taking into account both the general specifications and implementation requirements, through the multiple-objectives methodology. This approach has several advantages over the conventional one:

1. There is no need to find an intermediate Laplace transfer function for the compensator.

2. Since we are applying a "blind" search technique, no knowledge derived from design techniques conceived by humans needs to be supplied to the system.

3. Instead of sampling standard circuit topologies, novel implementations are also investigated. These new circuit topologies may have advantageous features in terms of number of components, dissipation, noise, and fault-tolerance aspects, compared to traditional implementations.

Time domain specifications are the most important to be fulfilled by control systems. The performance characteristics of a control system are often characterized by its transient response to a unit step function. Among other specifications, the rise time (t_r), over-signal (M_p), and settling time (t_s) must be observed in the design of control systems. These statistics are graphically illustrated in Figure 8.2. In this application, only t_r and M_p are considered as objectives to the GA, since the settling time, t_s, is roughly proportional to the over-signal. On the other hand, it can be mathematically proven that the simultaneous minimization of t_r and M_p characterizes a problem of satisfying to contrasting objectives.

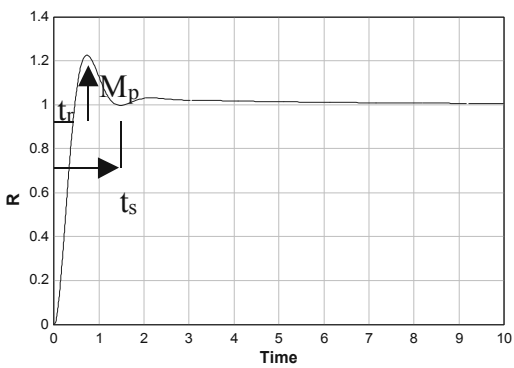


Figure 8.2 Hypothetical transient response of a system and illustration of the performance statistics t_r, t_s, and M_p.

This particular application focused upon single-loop feedback systems. In order to demonstrate the potential of the evolutionary methodology to fulfill the aspects presented in the last section, three experiments are described. In the first one, a circuit is synthesized to compensate a second-order system, considering the

following objectives to be minimized: Mean-Square Error (MSE) to the target response; rise time; maximum over-signal value; circuit offset; and dissipation. The minimization of the offset is necessary to set the correct circuit operating point. The second experiment included the same objectives, considering, though, a fourth-order system. The third case study considered also the minimization of the intrinsic noise. While in the first two cases, only bipolar transistors were used by the GA, the last case study allowed the use of MOS transistors as well. An additional experiment, which includes a fault tolerance requirement as an additional objective to the evolutionary process, is described at the end of this chapter.

8.2.1.3 First Experiment

This case study refers to the synthesis of a control circuit to compensate the following plant, G(s):

$$G(s) = \frac{4}{s(s + 0.5)} \tag{8.1}$$

The system is depicted in Figure 8.3.

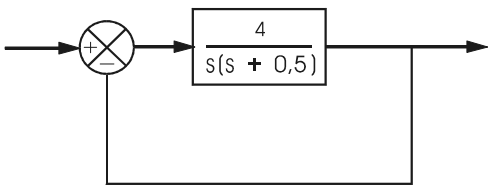


Figure 8.3 Second order plant to be compensated. Extracted from Ogata,[4] 1990.

Through analytical methods, the following compensating function, $G_c(s)$, was derived:[4]

$$G_c(s) = \frac{10 \cdot (2s + 1) \cdot (5s + 1)}{(0.1992\ s + 1) \cdot (80.19\ s + 1)} \tag{8.2}$$

Evolutionary computation is then applied as an automatic approach to compensate this system. Figure 8.4 depicts the schematic of the best circuit achieved in this experiment. Each circuit evaluation is accomplished by applying a single step voltage at the system input, and probing the plant output through a transient analysis. According to the representation used in this problem, three external points were defined:

1. The output of the comparator that subtracts the desired output value (the unit step voltage) and the plant output (see Figure 8.4)

2. The plant input, corresponding to the controller's output

3. A DC power supply of 5 Volts and the ground

Besides these four external points, the GA was also able to use four internal points to arrange the topology. The GA sampled a population of 40 individuals along 200 generations in this experiment.

Figure 8.5 compares the transient response of this analytical expression (equation 5) with the one of the control circuit achieved by the evolutionary system. The user's specifications or design plan were set to the following values: Maximum MSE of 10 (no unit); maximum rise time of 2 seconds; maximum over-signal of 1.2V; maximum offset of 1mV; and maximum dissipation of 100mW.

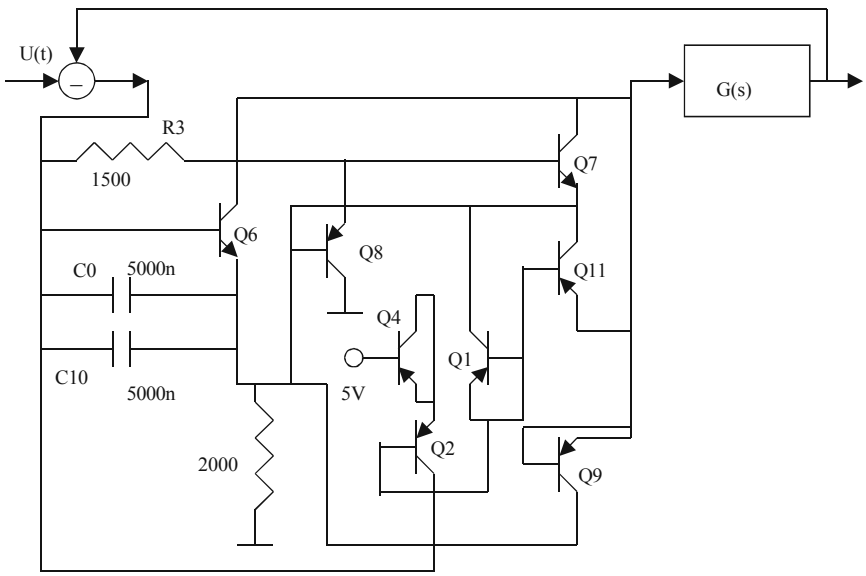


Figure 8.4 Schematic of the evolved control circuit for the first case study. U(t) is the plant's desired output (unit step).

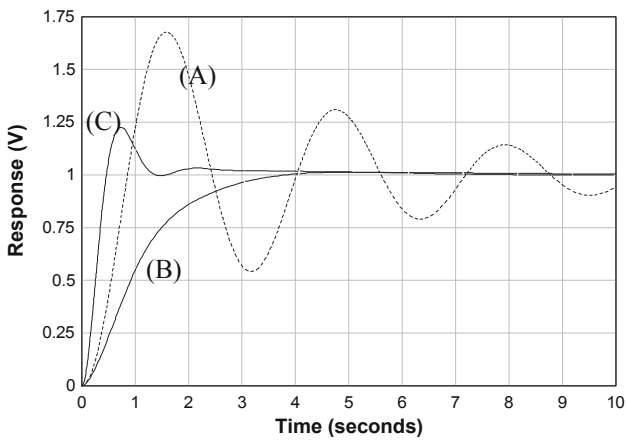


Figure 8.5 Comparison of the transient responses of: uncompensated system (A); evolved control circuit (B); and the analytical expression shown in equation (8.2) (C).

From the circuit shown in Figure 8.4, one can observe that the GA arrived at a very unusual arrangement of transistors. A total of 12 components were available for the GA to synthesize the circuit. When an error signal of 1V appears at the circuit's input, the following behavior is observed: transistors Q1, Q2, Q4, Q9, and Q11 work in the cut-off region; transistor Q6 and Q7 work in the saturation region; and transistor Q8 works in the linear region. In this case, the power dissipated by the circuit is about 10μW, much less than specified in the design plan. This dissipation is basically due to the current driven by the saturated transistors. The circuit offset is of the order of μV, also complying with the design plan.

From the comparison shown in Figure 8.5, we can verify that both the analytical and the evolved control perform better when compared to the uncompensated original system. The analytical control exhibits a lower rise time in comparison with the evolved control, but the former also exhibits a higher degree of oscillation.

We conclude that the GA was able to satisfy four out of five objectives. The rise time was the only specification kept above the required value, possibly due to the fact that a virtually null over-signal was achieved.

A second experiment refers to the synthesis of a control circuit to compensate the following plant:[3]

$$G(s) = \frac{1}{s(s+3)(s^2+s+4.25)} \tag{8.3}$$

This is a more complex case study if compared to the first one, not only because the plant is a fourth order system, but also because the compensator will have to provide more gain to achieve a good transient response. We used the same representation and objectives outlined in the first experiment, processing a similar number of individuals as well. The power supply voltage was decreased to 3V. Figure 8.6 depicts the schematic of the best circuit evolved in this experiment. Figure 8.7 compares the evolved system's transient response with the one of a plant compensated with a pure gain of magnitude 9 reported in the literature.[3] Figure 8.8 displays the objectives' specifications achieved by the best circuit at each generation of the GA.

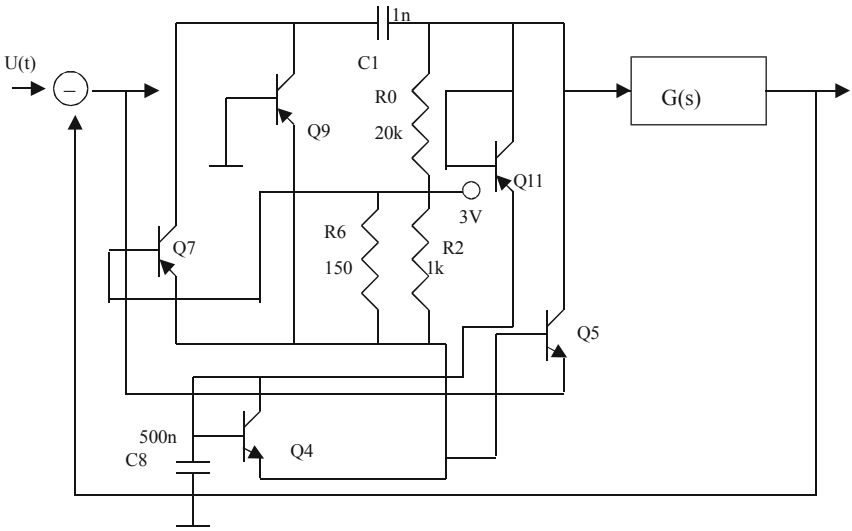


Figure 8.6 Best compensator evolved in the second experiment. U(t) is the plant desired output (single step).

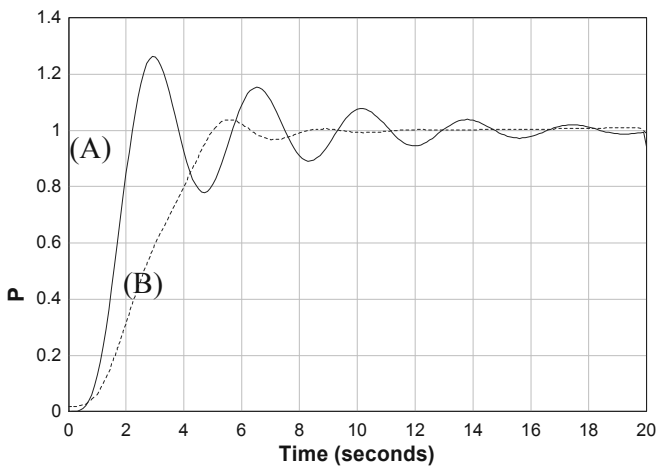


Figure 8.7 Comparison of the plant transient responses with a pure gain compensator (A) and with the evolved compensator (B).

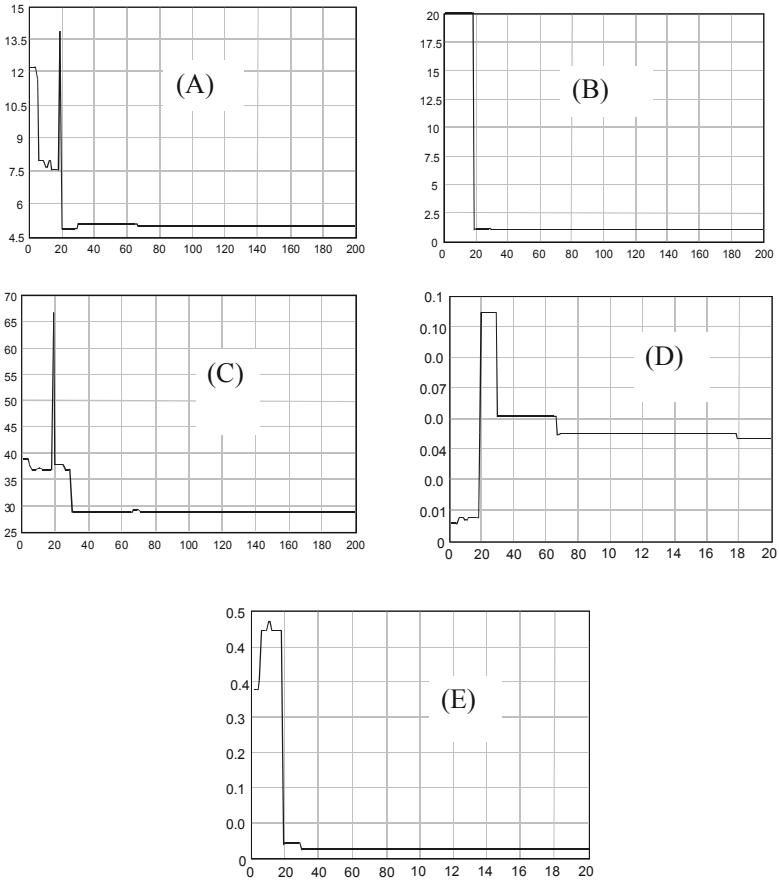


Figure 8. 8 Specifications attained by the best individual at each generation of the GA: (A) Rise time (seconds); (B) Over-signal (Volts); (C) Mean Square Error (MSE) to the desired response; (D) Dissipation (Watts); (E) Offset (Volts).

We can draw some conclusions about the circuit shown in Figure 8.6. This circuit delivers a voltage amplification of 20 dB (corresponding to the desired gain factor of 10), when the comparator's output is 0.2V. Transistor Q5 is the current amplifying device; transistors Q4, Q7, and Q11 work in the cut-off region; depending on the comparator's output, Q9 may work in the cutoff region or in the saturation region. In the latter, it will drain most of the current supplied by the DC power supply.

It can be seen, from Figure 8.7, that the settling time of the evolved compensator is much lower than the one of the pure gain compensator. Figure 8.8 illustrates the process of simultaneous minimization of many objectives, accomplished by the GA. From the graphs displayed in this figure, one can verify that, except for the dissipation, there is an *almost* monotonic decrease in the objectives' values. The dissipation starts at a very low value, increases to 0.5 W, and then is reduced to the level of 50mW.

Figure 8.9 depicts a conventionally designed compensator for this system, presented in the literature.[3] The transient response of this compensator is not shown in Figure 8.7. If we contrast this conventional circuit topology with the evolved one, we can observe that the evolved circuit uses less components, at the expense of being slower than the conventional one (rise time of 2.5 seconds exhibited by the conventional circuit, against around 4.5 seconds for the evolved circuit).

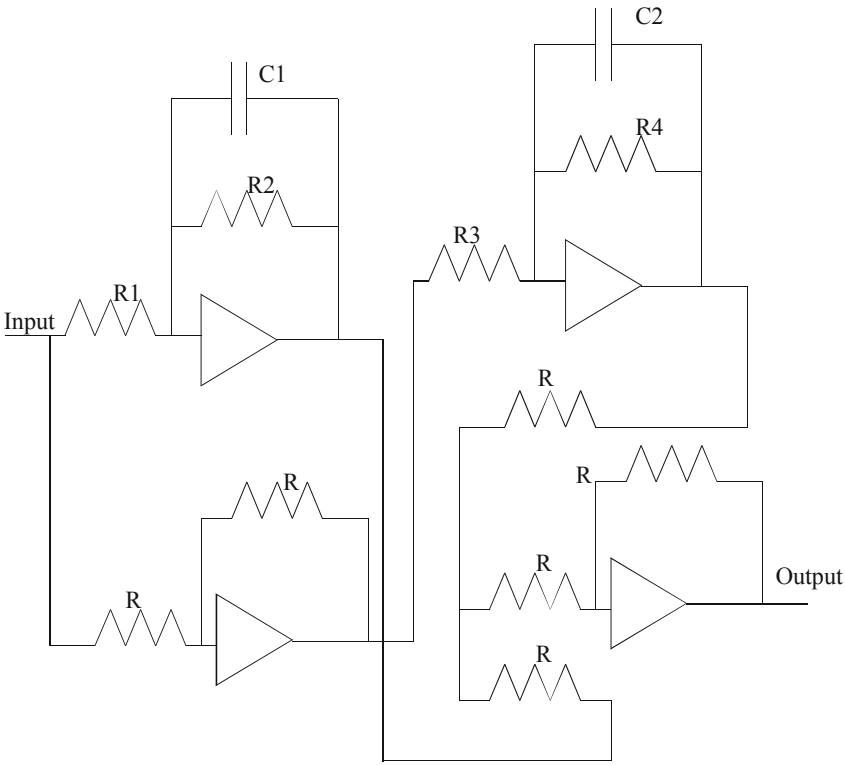


Figure 8.9 Active compensator, presented in the literature, for the fourth order system.

This last experiment involved the addition of a new objective, the minimization of the intrinsic noise of the circuit implementing the compensator. This experiment also attempts to evolve a BiCMOS circuit, by considering *n-channel and p-channel* MOS transistors as well. The MOS transistors have been simulated using SPICE's level 2 model, with channel's length and width equal to 20µm. The plant employed in the first experiment was also used in this case.

The GA execution sampled a population of 40 individuals along 40 generations. Figure 8.10 depicts the best circuit achieved. It can be seen that the GA used only MOS transistors to accomplish this task. Figure 8.11 compares the following responses: the controller of Figure 8.10; the uncompensated system; the compensator shown in equation (8.2); and the circuit evolved in the first experiment (Figure 8.4). Finally, Figure 8.12 plots the intrinsic noise generated by the circuits

evolved in the first and fourth experiments respectively.

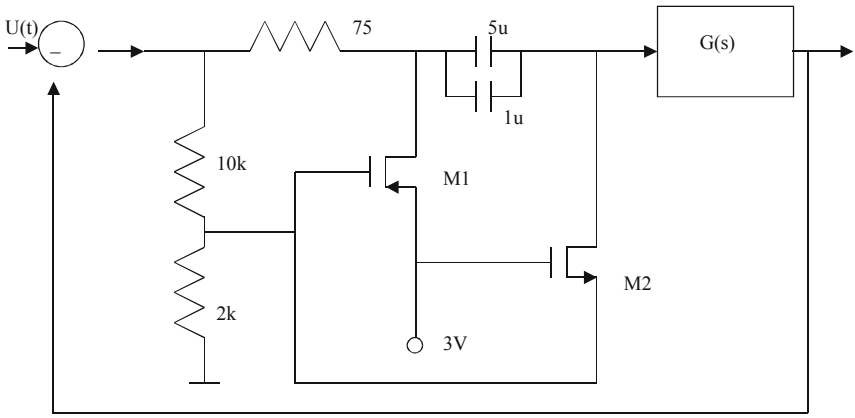


Figure 8.10 Best compensator evolved in the fourth experiment. U(t) is the plant desired output. G(s) is the second order plant.

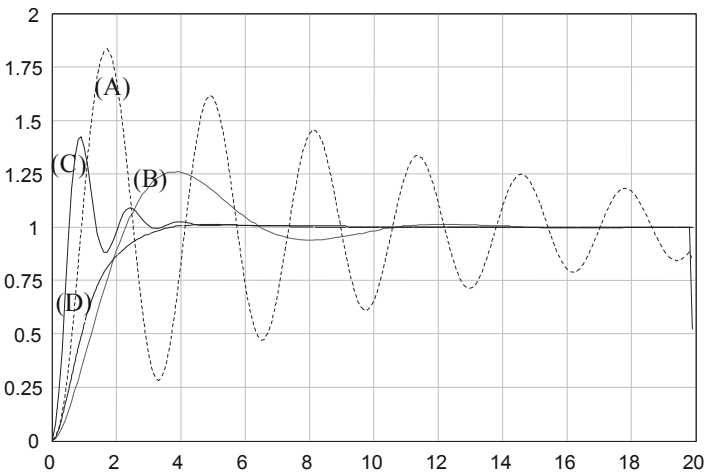


Figure 8.11 Comparison of the transient responses: (A) uncompensated system; (B) compensator of Figure 8.10; (C) Compensator of equation (8.2); (D) compensator of Figure 8.4.

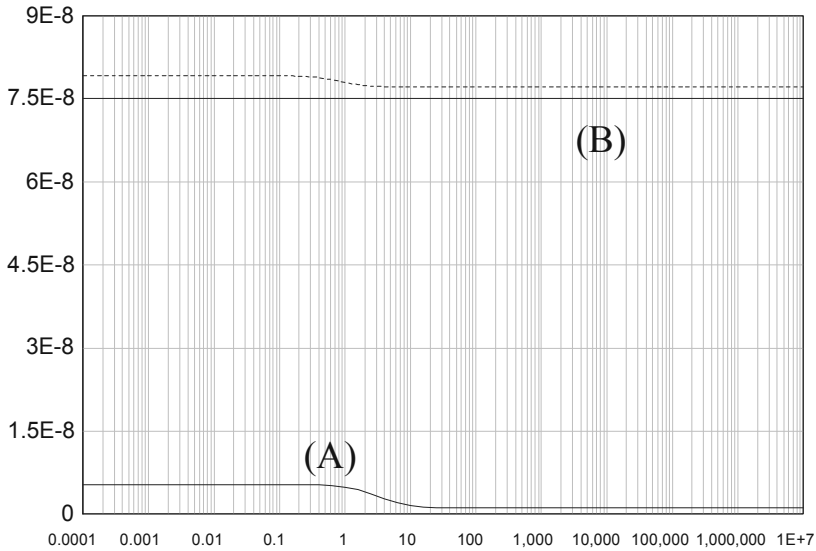


Figure 8.12 Output noise for the circuit evolved in the fourth experiment (A) and in the first experiment (B). Axis Y gives noise in $V/Hz^{0.5}$ and axis X gives the frequency in Hertz.

It can be verified, from the circuit's schematic, that the GA used only MOS transistors and resistors, not employing bipolar transistors in the design. Through the analysis of the circuit's operation, we found out that transistor M2 was simply working as a non-ideal switch, and it could be replaced by a resistance of 10k, connecting the output to the gate of M1. From Figure 8.11, we can see that the evolved controller is slower than the one presented in the literature, and it presents higher over-signal than the circuit evolved in the first experiment. Nonetheless, as shown in Figure 8.12, this circuit generates less noise than the one achieved in the first experiment.

8.2.1.5 Lessons Learned

The analysis of the results allows us to draw some conclusions about the advantages of evolvable control systems, compared to human designed systems. Since this application does not employ high level analog blocks, such as Operational Amplifiers, commonly used in electronic control systems, Evolutionary Computation is able to explore novel arrangements of analog components that go beyond the scope of conventional analog circuits. We also remark that the evolutionary system allows the designer to consider, with proper priority levels, diverse implementation requirements, such as the circuit dissipation, which is of great relevance in aerospace applications.

The evolved circuits presented the following advantages over conventional ones: lesser number of components and, as a consequence, less power consumption; and smaller over-signal values. However, the results were deceptive in terms of an

important performance statistic, the rise time. The evolved controllers were slower than the conventional ones. In order to solve this problem, priority levels should be changed in our multiple-objective methodology, and a higher priority must be conferred to the rise time. This approach can be particularly useful in the case of circuits that must adapt to changing environments. This objective can be attained through the use of reconfigurable analog chips to implement the evolved circuits: as the environment conditions change (in this case, the plant coefficients), new circuits may be evolved to cope with the specifications, with minimum human intervention. In the case of aerospace systems, for instance, there are promising applications of this method, such as in antenna positioning control, and in the control of rotor speed for energy storage systems in spacecrafts.

This methodology still needs to be improved to deliver more competitive results. Nevertheless, if successful, it will represent the achievement of a tool that is able to synthesize the final version of a control system (i.e., as an electronic circuit) from high level control specifications. The next application shows an alternative way to evolve controllers.

8.2.2 Evolution of PID controllers by Means of Genetic Programming

Genetic Programming is applied to the synthesis of PID (Proportional-Integral-Derivative) controllers. This class of controllers is very commonly used in industrial applications, and can be described by the following equation:

$$G_c(s) = K_p + K_i/s + K_d/s$$

where K_P is the proportional gain, K_i is the integral gain, and K_d is the derivative gain. These are the controller parameters. There are conventional techniques that allow us to find the parameters of a PID controller. Nevertheless, when the mathematical model of a plant is very complex, experimental techniques should be applied to find the parameters.[4]

This particular application refers to the automatic synthesis of PID controllers for plants that can be analytically described, like in the previous application. Next, the representation and fitness evaluation function are presented.

8.2.2.1 Representation

The program tree data structures of GP will represent block diagrams of the controllers. This representation allows the synthesis of both the topology and the parameter values for a controller. The function and terminal sets allow the mapping of a wide repertoire of block diagrams. Some of the possible functions are:

- The differentiator block *s*
- The integrator block *1/s*

- The lead block $1 + \tau s$
- The lag block $1/1+\tau s$
- The gain block
- Blocks to perform addition, subtraction, and multiplication of two signals
- The delay functions e^{-sT}

Therefore there is a fundamental difference between this high level representation based on building blocks and the low level representation used before. These functions allow the implementation of controllers that realize PID functionality. Figure 8.13 depicts an example of program tree.

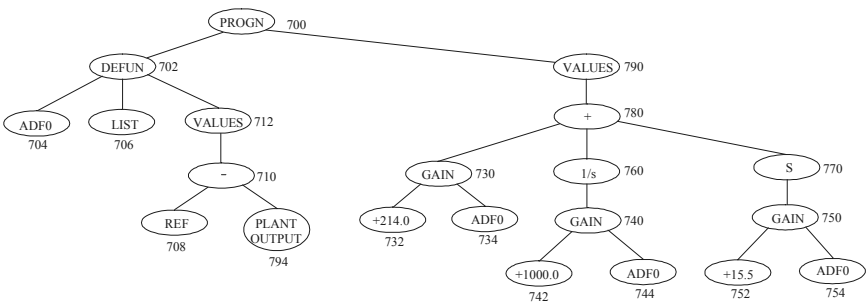


Figure 8.13 Program tree for control evolution. (Extracted from Reference 5)

Automatically Defined Functions (ADFs) are also used in this representation. The concept of ADFs have been defined previously in Chapter 2. In this particular class of problems they provide a mechanism to represent takeoff points that are disseminated to other points in the block diagram, and also to create internal feedback within the controller.

In the same way as when GP is applied to the evolution of electric circuits, numerical values need to be established for some blocks, such as the gain, lead, and lag. Arithmetic performing sub-trees can be utilized to calculate the numerical values for these blocks in the same way they were used to define component values in applications described in the previous chapter (evolution of analog circuits). A second approach also used in the particular application is the one of assigning perturbable numerical values to the blocks. This is described in Koza[5] (2000):

> *... These perturbable numerical values are changed during the run (unlike the constant numerical terminals of the first approach). In the initial random generation, each perturbable numerical value is set, individually and separately, to a random*

value in a chosen range (e.g., between +5.0 and –5.0). In later generations, the perturbable numerical value may be perturbed by a relatively small amount determined probabilistically by a Gaussian probability distribution.

The terminal set includes, but is not restricted to, the following signals:

- Reference signal for the controller
- Plant output
- Controller output

8.2.2.2 Fitness Evaluation Function

SPICE is again used to simulate the controller and the plant. SPICE is versatile to simulate not only electronic circuits, but also the block diagrams that define the controller in this application.

The fitness evaluation function considers both time and frequency domain aspects of the controller response. In the case of the analysis in the time domain, the fitness rewards the speed of the controller, robustness to variations of the plant's parameters, and overshoot avoidance.

In a particular case study, the following plant transfer function is considered:

$$G(s) = K/(1 + \tau s)^2$$

This is a two-lag plant. The objective is to create topology and parameter values for a controller such that the plant output achieves the level of the reference signal so as to minimize the integral of the time-weighted absolute error, such that the overshoot in response to the step input is less than 2%, and such that the controller is robust in the face of significant variation in the plant's internal gain, K, and the plant's time constant τ. For this case, the fitness was computed using different values for the internal gain K (1.0 and 2.0) and for the plant's time constant τ (0.5 and 1.0).

In addition to the time domain analysis, a frequency domain analysis is also performed in this particular application. According to Koza[5], this term of the fitness evaluation function is designed to constrain the frequency of the control variable to avoid high frequencies that could damage the components of the plant. A SPICE AC analysis is performed: a gain of –3dB is required from 0 until 100Hz, with a linear roll-off until a gain of –83dB is observed at 10,000Hz.

Summarizing, 10 SPICE simulations are necessary to evaluate each controller:

- *Eight* time domain based analyses, in which the weighted error to the desired response is computed. This number of analysis stems from the fact that the simulations combined two different values for K, two different values for τ, and also two different values for the height of the reference signal (1 micro-volt and 1 volt).

- *One* transient analysis that verifies the controller stability when presented to a spiked reference signal. The plant output should not exceed a particular boundary value in the presence of this spiked reference signal.

- *One* frequency analysis measuring the reasonableness of the controller's frequency response.

8.2.2.3 Results

After evaluating around 10^6 individuals, consuming 44.5 hours of computer time on a 66-nodes parallel computer system, a solution was found. The genetically evolved controller is shown in Figure 8.14.

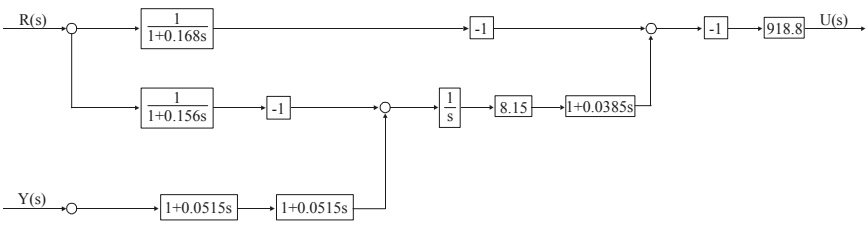


Figure 8.14 Genetically evolved controller through GP (extracted from Koza[5] (2000)).

Compared to a conventional controller to the same plant, the genetically evolved controller is better in terms of rise time, settling time, integral of time-weighted absolute error, and also at suppressing the effects of disturbance at the plant input. The basic difference between the evolved and the conventional controllers is that the former employs a second derivative processing block, not present in the latter. According to Koza[5] and fellows from the Genetic Programming Inc., the use of this derivative term was appropriate in this context. However, in a practical application, derivative terms may amplify high frequency noise effects.

8.2.3 Comparison

We can draw a comparison between the two applications described (GA and GP). The advantages of the GP-based evolution of controls systems were the following one:

- Some important aspects, such as plant disturbance, were also considered, increasing the robustness of the evolved solution.

- Competitive performance with conventional designs.

Nonetheless, we observe the following disadvantages:

- Need of powerful computer resources to accomplish evolution.

- The solution of the problem is given in the form of a block diagram, and not in the form of an electronic circuit. Therefore, hardware implementation aspects of the controller are not considered.

In order to achieve competitive and novel control circuits, a combination of the two approaches should be attempted in the future. This new approach should have the following features:

- Low level representation, so that the final product of evolution is an electronic circuit as in the GA-based application. The low level representation increases the chances of finding novel solutions to standard problems. Other low-level representations, such as the circuit constructing tree should be attempted.

- Reducing the number of SPICE evaluations per individual, particularly the ones based on transient analysis that are extremely time consuming. The plant disturbances should then be considered in a more efficient way than the one described in the last application.

- Tuning the multi-objective evaluation function to provide a better balance between rise time and overshoot.

8.3 EVOLUTION OF ANTENNAS

The concepts of circuit evolution can be extended to the evolution of antennas. As a matter of fact, recent results achieved in this area show that artificial evolution is able to synthesize antennas competitive or superior to the human designed ones. We divide this topic into the following sections: overview of antennas; evolution of antennas through GAs; and evolution of antennas through GP.

8.3.1 Overview of Antennas

This section briefly summarizes the concepts of antenna design, based on the introduction to antenna design found in Linden and Altshuler[6] (1999). An antenna is a device for receiving and transmitting electromagnetic waves. They can receive

an electromagnetic wave and transform it into a signal to a transmission line, or they can transform a signal from a transmission line into an electromagnetic wave that is then propagated in free space.[7] Some antennas are directional, receiving signals primarily in a specified direction. They may operate over a narrow or wide band of signals.

A *Wire Antenna* is a combination of a number of wires that are much longer than their width. Wire antennas are usually fed by a signal with a co-axial or two-wire line. Two important quantities in antenna design are the *gain* and the *directivity*. The directivity is the ratio of power density being transmitted by the antenna in a particular direction to the average of power density being transmitted to all directions. The gain is the directivity multiplied by the ratio of power radiated to the power input, taking into account the losses due to the resistance of the antenna. The *gain pattern* shows the proportion of power an antenna transmits in a particular direction, plotting the gain magnitude versus angle. The *antenna beamwidth* refers to the useful angle span, constituting the main lobe of the antenna. Secondary lobes are called *sidelobes*. The designer tries to maximize the main lobe and minimize the sidelobes.

The bandwidth is the useful range of frequencies for an antenna. For an antenna operating at 2GHz, a bandwidth of 3% means that the antenna would operate from 1.97GHz to 2.03GHz.[6] The bandwidth should be as large as possible.

The *Voltage Standing Wave Ratio (VSWR)* quantifies the match between an antenna and the device connected to it. A mismatch will prevent power from flowing to and in the antenna. The lower the VSWR, the better is the match between the antenna and the device. VSWR is given by the following equation:

$$\mathbf{VSWR = (1 + |R|/1 - |R|).}$$

R is the reflection coefficient computed by:

$$R = (Z - Zo)/(Z + Zo)$$

where Z is the impedance at the voltage source and Zo is the characteristic impedance of the transmission line feeding the antenna.

The *polarization* refers to the orientation of the electromagnetic wave. They can be *linearly polarized* if the electric field component oscillates in a single direction, circular or elliptically polarized if the electric field component is rotating (the magnetic field orientation is determined for each kind of polarization by the right-hand rule). Antennas are designed according to the particular wave polarization they need to pick up.

8.3.2 Evolution of Antennas Through GAs

8.3.2.1 Problem Specification

This application was developed by Linden and Altshuler[6] (1999). The objective was the design of an antenna to search for primeval hydrogen: the band of interest was the one between 219MHz and 251MHz. The most important design goal was to have the sidelobes and backlobes at least 25dB down in the region from $70^o < \phi < 290^o$, where f is the azimuth angle (plane XY of the antenna). The reason is the interference from surrounding radios and TV towers. The gain should be as high as possible and the VSWR should be under 3.0.

8.3.2.2 Representation

It was chosen to use GAs to optimize a Yagi type antenna, which is normally used for high gain and narrowband applications. Figure 8.15 shows the structure of a Yagi antenna:

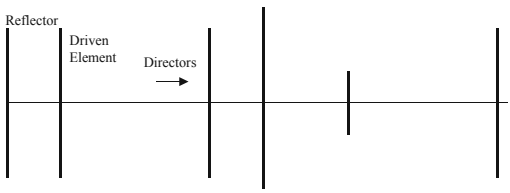


Figure 8.15 – Yagi Antenna (extracted from Linden and Altshuler[6] (1999)).

The Yagi antenna consists of a series of parallel wires. One of the wires is excited;

a second one stays behind this driven element, and it is called the reflector. The other ones are called directors. The driven element is a vertical linear element positioned along the Y axis, whose midpoint is connected to the transmission line. The other elements of the antenna are not connected to the transmission line; instead, their currents are induced parasitically by mutual coupling.[7] The directors are typically spaced unequally, shorter than the driven element and of different lengths. The Yagi antenna is linearly polarized.

The GA tries to find good values for the length of *N* wires, the spacing between each set of two elements, and the diameter of the wire. This provides a total of 2N variables. In this case, the number of wires was specified to be 14. A real-valued chromosome represented these quantities. The population size was of 175 individuals.

8.3.2.3 Fitness Evaluation Function

In order to evaluate the performance of each antenna, they were simulated using the Numerical Electromagnetics Code, version 2 (NEC2). This problem is intrinsically multi-objective (as all the problems in !). The fitness evaluation function was a weighted sum of the objectives that needed to be satisfied, i.e.:

- Maximizing lowest gain of all frequencies at angle 0^o
- Minimizing sidelobe levels in the region $70^o < \phi < 290^o$; they must be at least 25dD lower than the gain at 0^o
- Minimizing VSVW for all frequencies

The GA was able to find an antenna that met all the criteria. The so-called Genetic Yagi antenna was very different from the conventional ones, in that the average spacing between the directors was smaller comparing to the conventional case. This genetic antenna was manufactured, with measured parameters in agreement with simulated ones.

8.3.3 Evolution of Antennas Through GP

Yagi antennas are also the focus of this case study, in which Genetic Programming is used to rediscover essential characteristics of this class of antennas.[7]

8.3.3.1 Problem Specification

The problem to be tackled is to synthesize a planar symmetric antenna composed of wires of half millimeter radius that:

- Has maximum gain along the X-axis over a range of frequencies from 424MHz to 440MHz
- VSWR < 3.0
- Fits into the bounding rectangle of 0.4m height and 2.65m width
- Is excited by one single voltage source connected to the driven element by a transmission line whose characteristic impedance is 50 Ω

8.3.3.2 Representation

Contrasting to the previous application, the number of directors and reflectors are not pre-specified. Moreover, the driven element, directors, and reflectors are not necessarily straight wires and also may not be arranged in parallel. This stems from the more flexible representation offered by GP compared to classical GAs.

The GP trees encode a two-dimensional drawing called turtle geometry. The turtle moves on a plane and may or may not deposit wire when it moves. The functions used in the program trees determine the movement of the "turtle," such as TURN-RIGHT, TURN-LEFT, etc. The main terminals are called HALF-MM-WIRE and

NOWIRE, which cause the turtle to deposit or not wire respectively. Also real numbers are used to determine turning angles, distance to move, etc.

Therefore, there are no constraints to the number or shape of wires. The only constraint is to delete any piece of metal that stays outside the bounding rectangle defined above. Also, since a symmetric antenna is required, every piece of metal created above the X axis is reflected across this axis.

8.3.3.3 Fitness Evaluation Function

As in the previous case the NEC simulator was used. The fitness was a combination of the gain and VSWR, in a similar way to the one done in the previous application. In this particular case the NEC simulator was instructed to compute the Fairfield radiation pattern at $\theta = 90$ (angle from the positive Z-axis to the X-Y plane) and $\phi = 0$ (angle from the positive X-axis to the positive Y-axis). The value of the antenna gain is the magnitude in decibels of the farfield radiation pattern.

8.3.3.4 Results

As in the case of electronic circuit evolution, GP required a much larger number of individuals, 500,000 in this case. The GP was implemented in parallel in a home built parallel cluster computer system consisting of 1,000 350MHz Pentium II processors. Subpopulations of 500 individuals were processed by each of the 1,000 nodes. Twenty-two hours of computation time were necessary to achieve the best individual at generation 90.

The best result was an antenna with the essential features of the Yagi antenna, i.e., one reflector, straight line driven element, and multiple parasitically coupled straight line directors (total of 33). The performance was the same for a conventionally designed Yagi antenna in terms of gain, VSWR, and length along the X axis.

Even though the representation did not impose constraints, GP was able to "rediscover" the Yagi design.

8.4 FAULT TOLERANCE AND EVOLUTIONARY ELECTRONICS

Fault tolerance is a very important requirement for electronic systems, particularly the ones that operate in harsh environments, where the replacement of faulty components may take too much time or even be completely impossible. A typical example comes from the area of space exploration, where electronic circuits must "survive" in unknown environments. A common strategy to achieve fault tolerance is through the use of redundant hardware to replace faulty circuits. Artificial evolution, on the other hand, may achieve fault-tolerant circuits in different ways. In order to understand the mechanisms in which artificial evolution produce fault tolerant hardware, we can use some aspects of natural evolution as metaphors. Adaptation through natural selection and mutation is an evolutionary mechanism that confers resistance to bacteria population against antibiotics. Similarly, the

genetic variation of the human species also helps the species survivability against micro-organisms' infections. *Graceful degradation* is another metaphor, being observed in natural systems but missing in human designed systems. The best example of graceful degradation comes from the human nervous system, where the loss of neurons affects the brain performance in a very gradual and slow way. We can contrast with human designed electronics, where one simple fault usually causes the circuit to work completely out of the specs. The applications here described play with the ideas of mutation, genetic variation, and graceful degradation in artificial systems.

One of the first works to address the issue of exploration of fault tolerance using evolutionary techniques was the one of Thompson[8] (1996). In his work, he defines two ways to achieve fault tolerance through evolution, one implicit and the other explicit. The former is based on the assumption that evolution can provide some degree of fault tolerance for free using the population dynamics. The latter consists of explicitly incorporating fault tolerance requirements into the fitness evaluation function.

Thompson demonstrated this concept by evolving the contents of a RAM that controls a robot in a wall avoidance task. The RAM consisted of 32 bits that could be mapped in a straightforward way using binary chromosomes. After 85 generations a solution to this task was achieved, and 32 faults were applied to the best individual by respectively switching each of its 32 bits. The worst fault, the one that most deteriorated the individual's performance, was selected. Another generation was performed with the worst being applied to all individuals during the fitness evaluation function. A new "worst fault" was determined for the best individual of this generation, and applied subsequently in the next generation. The process was then repeated each generation. At generation 204 an individual that was tolerant to all the 32 possible faults was achieved.

Thompson's ideas were further developed and tested by other researchers. In this section we describe three applications, one using the implicit approach, the second using the explicit approach, and the third one comparing both approaches.

8.4.1 Implicit Fault Tolerance

Implicit fault tolerance, as originally called by Dr. Adrian Thompson, is also named population based fault tolerance. This idea is based on the assumption that artificial evolution provides some degree of fault tolerance for free, i.e., an inherent quality for the circuits produced by this design process. The work of Layzell and Thompson[9] (2000) performs a deeper investigation on the phenomenon of Population Fault Tolerance (PFT). They define PFT as the potential for a population of evolved circuits to contain an individual which adequately performs a task in the presence of a fault that renders the previously best individual useless. This idea is based on the first experiments performed by Thompson.

The experimental framework consisted of the Evolvable Motherboard, which was described in Chapter 7. The Evolvable Motherboard is based on plug-in components that are interconnected through analog switches. Due to the fact that components

can be easily removed from the board, it is ideal for fault tolerance experiments.

In this set of experiments, only bipolar transistors (PNP and NPN) were employed. In one of the experiments, they targeted the evolution of oscillators, using the following fitness equation:

$$F_{osc} = <a> + f_{min}/f_{max}(2f_{target} - |f_{target} - <f>|)$$

$$F_{osc} = <a> \text{ if } <f> < 60\text{Hz}.$$

where <a> and <f> are respectively the average, over 20 samples, of the amplitude and the frequency of the output signal. f_{min} and f_{max} represent respectively the minimum and maximum frequencies measured during the sample period (2ms), and f_{target} is the target frequency. The ratio of minimum and maximum frequencies serves to penalize individuals with varying output frequency. It is interesting to note that the analog hardware used to evaluate the oscillators included a frequency to voltage converter. This is a way to prevent aliasing effects that could be caused by frequencies higher than the A/D card sampling rate.

The experiment was successful; oscillators were evolved using a frequency of 25kHz as the target, with an error around 1%.

The basic strategy to evaluate fault tolerance consisted of removing each transistor (individually) from the best circuit achieved by evolution. Three situations may then occur: (1) the best individual continues to work normally without the removed component; (2) the best individual and all the other individuals of the population do not work after the component is removed; (3) the best individual does not work, but another individual in the population does work properly when the component is removed. The third situation characterizes PFT. After 20 runs of the oscillator experiment, Layzell and Thompson observed many cases in which PFT was taking place.

After having established the existence of PFT, they performed an investigation to understand the mechanisms producing it. They arrived at the following conclusions:

- The transistors whose removal produced PFT were active in the oscillator circuits. Therefore, they could *discard* the hypothesis that the PFT transistors were only acting as resistances, being of secondary importance to the circuit.

- Through additional experimentation, they found out that the transistors whose removal manifested PFT were the ones added in the latest stages of the GA. On the other hand, transistors adopted by earliest solutions in the evolutionary process are less likely to induce PFT.

The second conclusion shows that the evolutionary history can be used as a means to predict fault tolerant behavior. More importantly, their first conclusion states that PFT does not arise from “almost redundant” components added along the evolutionary process.

Just like in the natural case, the existence of a population with some degree of genetic variation helps the "survivability of the species" under occurrences that renders the fittest individuals useless to perform a particular task.

8.4.2 Explicit Fault Tolerance

In order to show the concept of explicit fault tolerance and graceful degradation, the evolution of electronic control circuits using low-level granularity, as previously described in this chapter, serves as a case study.

The fitness evaluation function assessed the circuit behavior in the presence of component faults, by removing the component from the circuit, i.e., a faulty component is modeled by an open-circuit. This procedure is similar to the one utilized by Layzell and Thompson[9] (2000), but here SPICE simulations were used to evaluate each circuit, simplifying the procedure of applying faults to the circuits. Faults are simulated for each individual along the evolutionary process.

An important question arises: how to incorporate fault tolerance properties into the circuit evaluation? The experiments were carried out through the inclusion of another objective into the GA. Evolution was accomplished in the following way: given a circuit with *n* components, we had it simulated *n* times, each of which with a particular component removed from the topology. The average performance of the circuit, calculated over its *n* faulty versions and including all the objectives, was taken. The quadratic distance between this average performance vector and the vector of desired objectives' values was then computed. The lower this value, the higher is the circuit's robustness to this kind of fault. This real value was included as an additional objective using the Energy Minimization multi-objective technique.

The main drawback of this approach is the increase in the experiment's time. The GA processed 80 individuals along 80 generations. As each circuit was made up of eight components, (8x80x80) additional evaluations were performed, apart from the (80 x 80) evaluations of the non-faulty versions of the circuits. A total of 57,600 evaluations were performed, and the experiment lasted around 48 hours in a PC. The fact that transient analysis is used to evaluate the circuits contributed to increase in the evolution time.

The objective of the experiment was to compensate the plant shown in Figure 8.3. The other objectives included were the rise time of the circuit, overshoot, and power dissipation. Figure 8.16 depicts the best circuit achieved in this experiment. Figure 8.17 compares the transient responses of the following systems: *(A) uncompensated plant; (B) non-faulty compensator; (C) compensator without Q1 (see Figure 8.16); (D) compensator without Q2; (E) compensator without Q3; (F) compensator without Q4; (G) compensator without R5; (H) compensator without Q6; (I) compensator without Q7.*

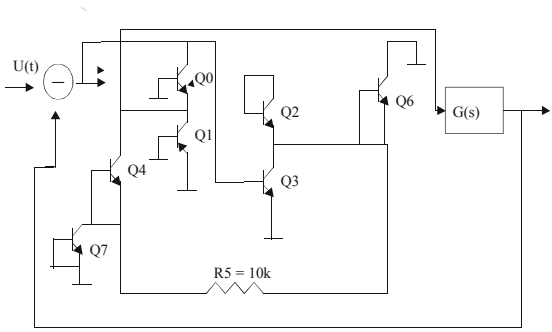


Figure 8.16 Best compensator evolved in the third experiment. U(t) is the plant's desired output. G(s) is the second order plant.

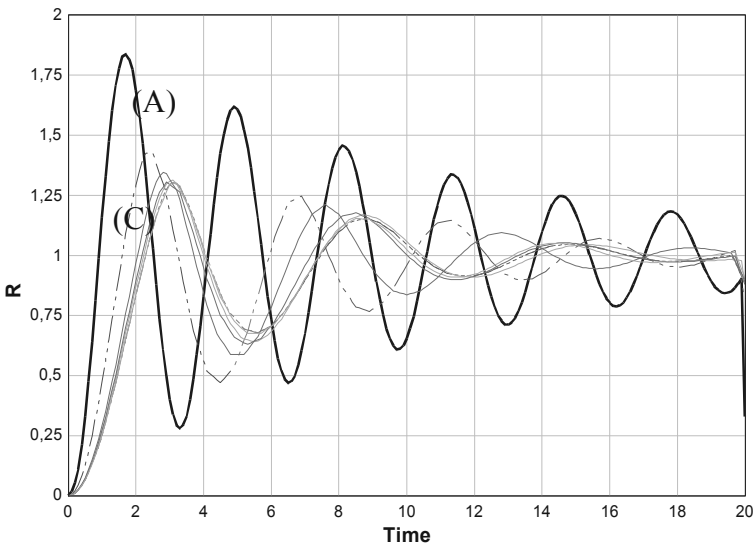


Figure 8.17 Transient responses achieved in the third experiment. (A) represents the response of the uncompensated system; (C) is the plant response without transistor Q1; the other curves are not labeled because they display a very similar shape.

The transistor Q0 was the only component that could not be removed from the circuit, since the circuit would completely stop working. From the circuit schematic, it can be verified that Q2 is redundant to the circuit, and, when taking Q4 away, the components Q7 and R5 are also removed (floating components). From Figure 8.17 it can be observed that the faulty circuits' versions still exhibit some degree of compensation.

This experiment did not produce a high performance circuit; however, the evolved circuit presents some interesting properties. The inclusion of a fault tolerance term into the fitness evaluation function drove the GA to a circuit that achieves some degree of compensation even with its components removed. Only one component is essential to the circuit behavior. This property resembles the graceful degradation,

in the sense the circuit is still able to present some functionality with components removed. This is different from the effect observed by Layzell and Thompson. In the latter, another individual in the population recovered functionality; in this case, the same individual keeps the functionality under component removal. This application is still very limited in the sense that human designed controllers outperform the evolved ones. However, two new concepts were tested, the explicit integration of fault tolerance into the fitness evaluation function and the graceful degradation. In order to further develop this idea, more computational power is needed to run the GA for more generations. Another approach is to test this concept for applications that are not so computationally expensive as in the case of the evolution of control circuits.

8.4.3 A Comparison between Implicit and Explicit Fault Tolerance Techniques

Keymeulen[10] et al. (2000) used the Field Programmable Transistor Array (discussed in the last chapter) to perform fault tolerance experiments in real hardware. An XNOR gate has been evolved with fault tolerance requirements being introduced using implicit and explicit mechanisms.

Figure 8.18 illustrates the best circuit achieved in one particular experiment. The two digital inputs are *In1* and *In2*, and the circuit output is named *Out* in this figure. In this case, two FPTA cells (refer to Chapter 7) were cascaded, and the output is taken from the second cell. Figure 8.18 also illustrates the six faults that were applied to the best circuit, by opening (fault 1, fault 2, and fault 4) or closing (fault 0, fault 3, and fault 6) six switches.

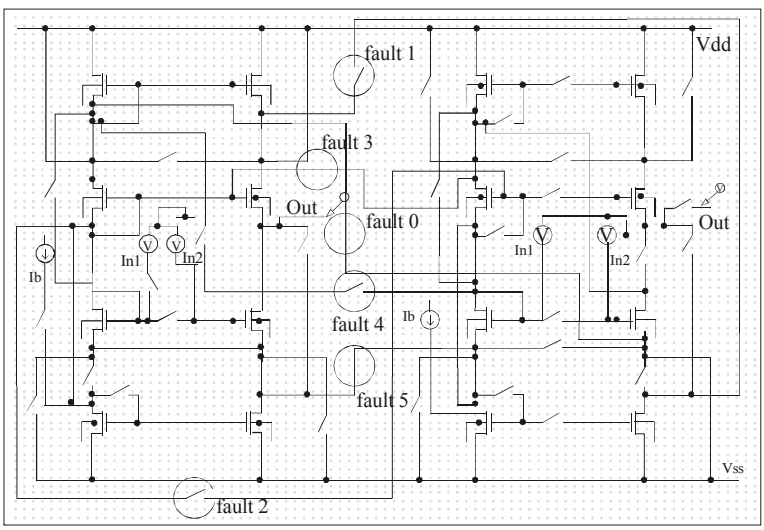


Figure 8.18 Evolution of a Fault Tolerant XNOR gate using the FPTA reconfigurable chip.

After applying all faults to the best individual of the population, a deterioration in

the circuit behavior was observed for at least some of the six faults. The following step was the application of the faults to the other individuals of the population (also called mutants). Usually, at least one individual in the population could be found that performed better than the best individual in the presence of faults.

In this particular case, the individuals of the final population had very similar genetic material, since the population was "almost converged" to a particular solution. Only a few mutations needed to be performed to go from the best individual to other individuals in the population. However, drawing an analogy with the natural case, this small genetic variance was enough to provide another individual in the population that would perform better in the presence of faults.

The explicit fault tolerance mechanism consisted of evaluating each individual without and with the faults along the GA, similar to the procedure applied to the evolution of the controller. However, in this particular case, the fitness was given by the average behavior of the circuit with and without the faults. In the case of the controller, on the other hand, fault tolerance was another objective included in the Energy Minimization method.

As in the case of Thompson's experiments, the capacity to recover under the presence of faults was also tested in this case. After the population had converged to a particular solution, a fault that was never seen by the circuit was applied to all individuals in the population. The GA was re-initiated and after some generations another solution that could cope with the fault was found. This shows that evolution can explore the properties of the faulty components to achieve a new circuit that is functionally correct.

The following conclusions were derived from this experiment:

- The FPTA architecture was versatile enough to quickly evolve an XNOR gate thoroughly compliant to the specifications.

- The implicit technique based on the population dynamics outperformed the explicit technique, based on the fitness evaluation function.

The second conclusion shows that, for this particular case, the "free" fault tolerance behavior provided by evolution through the mutants performed better than the circuit evolved through fitness induced fault tolerance.

8.5 SUMMARY

This chapter illustrated applications of in different domains, i.e., evolution of cell phone receivers, evolution of control circuits, evolution of antennas, and synthesis of fault tolerant circuits.

The evolution of receivers for cell phones shows that, even though being a recent area of research, is already delivering commercial products. It also demonstrated the importance of exploring novel technologies for circuit evolution, such as G_m filters.

Two approaches using different levels of representation were described for the evolution of control circuits. Whereas the high level representation presented better results, the low level representation seems more promising to tackle implementation requirements (low power, low-noise and fault tolerant circuits) and also to produce novel solutions.

It has also been shown in this chapter the potential of Evolutionary Computation to rediscover traditional antenna designs beginning from scratch, and also the evolution of novel designs that meet specifications for real world problems.

Finally, this chapter also described as a mechanism to produce fault tolerant circuits. Future design constraints characterized by fully automated requirements together with availability of a small time for system recovery renders a promising tool for the design of fault tolerant systems.

REFERENCES

[1] Murakawa, M., Yoshizawa, S., Adachi, T., Suzuki, S., Takasuka, K., Iwata, M., and Higuchi, T., Analogue EHW chip for intermediate frequency filters, in *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, Lausanne, Switzerland, September 23-26, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 134.

[2] Johns, D. A. and Martin, K., *Analog Integrated Circuit Design*, John Wiley & Sons, New York, 1997.

[3] Fortmann, T. E. and Hitz, K. L., *An Introduction to Linear Control Systems*, Mendel, J. M., Ed., Marcel Dekker, New York, 1977.

[4] Ogata, K., *Modern Control Engineering*, Prentice-Hall, Eaglewoof Cliffs, NJ, 2nd ed., 1990.

[5] Koza, J. R., Keane, M. A., Yu, J., Bennett III, F. H., and Mydlowec, W., Automatic creation of human-competitive programs and controllers by means of genetic programming, in *Genetic Programming and Evolvable Machines*, Kluwer Academic Publishers, 1, 121-164, 2000.

[6] Linden, S. D.and Altshuler, E. E., Evolving wire antennas using gGenetic algorithms: a review, in *The First NASA/DoD Workshop on Evolvable Hardware*, IEEE Press, Pasadena, CA, 1999, 225.

[7] Comisky, W., Koza, J. R., and Yu., J., Automatic synthesis of a wire antenna using genetic programming, in *Proc. of the 2000 Genetic and Evolutionary Computation Conference*, Late Breaking Papers, Las Vegas, 2000, 179.

[8] Thompson, A., Evolutionary techniques for fault tolerance, in *Proceedings of the UKACC International Conference on Control, (CONTROL'96)*, IEE Conference Publication No. 427, 1996, 693.

[9] Layzell, P. and Thompson, A., Understanding inherent qualities of evolved circuits: evolutionary history as a predictor of fault tolerance, in *ICES2000, International Conference on Evolvable Systems: From Biology to Hardware, LNCS 1801*, Miller, J., Thompson, A., Thomson, P., and Fogarty, T. C., Eds., Edinburgh, 2000, 133.

[10] Keymeulen, D., Stoica, A., and Zebulum, R., Fault-tolerant evolvable hardware using field programmable transistor arrays, in *IEEE Transactions on Reliability, Special Section on Fault-Tolerant VLSI Systems*, vol. 49, No. 3, IEEE Press, 2000.

CHAPTER 9

Conclusions

This chapter summarizes the main ideas expressed in this book. By contrasting evolutionary and conventional design, we show that the former is a promising alternative for modern electronics. We also describe the state of development of programmable chips, a key technology for Evolvable Hardware. We remark that Evolutionary Computation may have an impact in the relevance of two branches of electronics: analog and digital. Finally, we point out that the evolutionary design of fault tolerance systems is one of the most important topics of research in

9.1 EVOLUTIONARY AND CONVENTIONAL DESIGN

Evolutionary design is based on a parallel search in a large space of possible solutions to a particular problem. This search is driven by the use of evolutionary inspired operators, such as selection, crossover, and mutation. Conventional design is based on mathematical formalism, heuristics, approximated physical models, and also on human experience and intuition.

Evolutionary systems have appeared in the 60's as a novel class of algorithms developed to solve problems in the area of engineering design. The results were still limited, due to the incipient state of development of computing systems. The advancing in speed and memory aspects of microprocessors systems, together with the compilation of the early evolutionary techniques into the classical Genetic Algorithms, made possible the boost in the applications of evolutionary techniques, mainly in the optimization domain.

Currently, evolutionary systems are again the focus of research in the area of engineering design. Particularly, this book offered the perspective of the automatic design of electronic circuits through evolutionary systems. The current state of the art in microprocessors' systems has enabled the sampling of larger search spaces by Genetic Algorithms, improving their performance in electronic design problems. In addition, modern issues of electronics design, such as the achievement of low-power and low-noise systems, point towards the use of multiple-objective evolutionary systems. Contrasting with standard techniques for circuit design, Evolutionary Algorithms can contemplate not only general performance requirements for electronic circuits, but also the hardware implementation requirements previously mentioned.

9.2 PROGRAMMABLE CIRCUITS

This book also discussed the topic of programmable circuits. Programmable digital chips, mainly FPGAs, have become very popular in the last years, due to their versatility and speed of reconfiguration. Particularly, they gave birth to the area of

Virtual Computation, by means of which the hardware can be optimized to the implementation of each particular algorithm.

On the other hand, the analog programmable chips are still in earlier stages of development compared to the FPGAs. The Field Programmable Analog Arrays are usually regular architectures consisting of a matrix of Configurable Analog Blocks (CABs), also called cells. They are often identical, being constituted by one Operational Amplifier (OpAmp) with programmable interconnections. The design of FPAAs faces many challenges, the most important ones being the bandwidth, switch resistances, accuracy, noise, and area. We now summarize current technologies being used to address these problems:

- **Switched Capacitor (SC) Technology** – This technology is used in the Motorola MPAA0020 FPAA. It confers advantages in terms of accuracy in frequency response. However, the bandwidth of the circuit is limited by the clock frequency. Another problem with SC is the lack of simulation accuracy using standard simulators.

- **Current Conveyors** – The use of current conveyors as FPAA cells, instead of OpAmps, allows the bandwidth to increase until about 10 MHz, compared to 1MHz observed in other models. Current conveyors are very similar to OpAmps and can also implement many different analog sub-systems. The reader can refer to.[2] for more details.

- **BiCMOS Technology** – The use of BiCMOS technology, as proposed in.[4] is another way to increase speed and bandwidth of the reconfigurable device.

- **PulseWidth Technologies** – This technology is proposed in.[3] Instead of representing signals as voltages or currents, digital pulses are used to represent discrete analog signals. This approach facilitates the interface with digital systems.

- **External Components** – In order to minimize the chip area, external capacitors and resistances can be used, instead of integrating them in silicon, as in the Zetex chip.[4] However, this approach reduces the versatility of the reconfigurable device.

- **Differential Architecture** – This strategy is followed by the Lattice FPAA design,[5] and it is effective to improve the performance in terms of input common mode rejection and dynamic range.

- **Antifuse Switches** – The advantage of using antifuse technology as switches is the fact that it can provide a lower resistance (around 20 Ω) compared to MOS based switches (around 1K), thereby increasing the device bandwidth. The drawback is the fact that antifuses are only one time programmable.

Both FPGAs and FPAAs are key technologies to *Evolvable Hardware (EHW)* applications. EHW is the branch of in which the circuit evolution is accomplished in hardware. This book contrasted this approach (also called intrinsic EHW) with the extrinsic approach, where circuit simulators are used for the evolution. The intrinsic approach offers the advantages of exploring unconventional silicon properties, which can lead to novel and more efficient engineering design solutions. Nevertheless, these solutions may suffer from *stability* and *robustness* problems. The stability problem stems from the fact that many hardware evolved solutions are transient solutions [Stoica], in the sense that they work for a limited amount of time until they reach a final state different from the desired behavior. The lack of robustness derives from the fact that evolution may be exploring physical properties particular of the reconfigurable chip being used, which may change even if we use another (theoretically) identical chip. Conversely, circuits evolved in a simulator may not work in practice due to the lack of accuracy of the device models.

A more complete discussion on issues of intrinsic and extrinsic evolution can be found in Zebulum et al.[1] (2000). Overcoming the problems in both intrinsic and extrinsic evolution is a current topic of research in . Finding methods that successfully solve these problems will be a definite step towards the use of evolved circuits in industry.

9.3 CONSEQUENCES ON ANALOG AND DIGITAL ELECTRONICS

Results achieved by researchers in strongly suggest that the impact of this area in analog electronics will be much higher than in digital electronics. This is due to many facts, such as: the search space generated by analog electronic design is more amenable for evolutionary techniques; analog design is more complex than digital design; analog design is not as methodological as digital design; and, finally, the fact that it is increasingly difficult to find experts in analog electronics.

Although the end of analog electronics, due to advances in digital technology, had been predicted years ago, this assumption turned out to be completely wrong. Since we live in an analog world, analog circuitry is still needed to provide the interface with digital chips. In addition, analog circuits have the advantages of being faster and less power consuming than digital circuits. The challenge is to find design techniques that can compensate the drawbacks of analog electronics, mainly the issue of component drift. Evolutionary computation may be an answer to this quest. We showed in this book how Genetic Algorithms can be used to design a wide variety of analog circuits, such as: amplifiers; filters; computational circuits; and control circuits. In many cases, artificial evolution was able to find designs that

equal or even outperform human design. Nonetheless, in order to deploy evolved analog circuits in the real world, the question of porting circuits evolved in simulation to actual hardware has still to be further investigated. The appearance of more accurate simulators and more versatile models of programmable analog circuits promises to eliminate this gap between simulation and real hardware.

However, it has also been shown in this book that is also a powerful tool for digital design. We observed that could find novel and more efficient circuits for the implementation of basic combinational functions, such as arithmetic circuits. Novel sequential circuit designs for the control of simple CPUs have also been shown in this book. The main promise of the evolutionary design of digital circuits is possibly in the synthesis of digital filters at the gate level, as discussed in Chapter 6.

9.4 FAULT TOLERANT SYSTEMS

Besides the issues of low-power and low-noise electronic design, Evolutionary Computation can also address the issue of fault-tolerant circuit design. Modern hardware systems should be able to keep performing satisfactorily even in the presence of one or more faults. This is very critical in the area of space applications, where the hardware is subject to harsh environments, and it must work for large periods of time without direct human maintenance. Generally speaking, industry will also benefit from a new class of robust hardware systems that can cope with faults, at least for some period of time, until they are replaced. Besides robustness, researchers also search for a new generation of self-repairing hardware, which is able to achieve the same performance as the one observed prior to a fault through self-reconfiguration. Evolvable Hardware is a key technology for the achievement of this new category of fault tolerant hardware.

We demonstrated in this book experiments targeting the evolutionary design of fault tolerant analog control circuits. Our approach consisted in including several conditions in the fitness evaluation function, an approach called fitness-based fault tolerance. We also discussed the so-called population-based fault tolerance, where other individuals in the population of electronics circuits are examined in the presence of faults. It has been verified that “mutants” could be often found in the population, responding better to some faults than the best individual for non-faulty conditions.

Biological organisms show a very high degree of robustness that would confer many benefits to artificial systems: graceful degradation to aging; an immunological system that provides an autonomous defense against external aggression; and also relative robustness against temperature variations. In the future we shall attempt to borrow more concepts from nature to try to achieve better artificial systems. Evolutionary Computation is one particular method that conveys ideas from the natural to the artificial world.

REFERENCES

[1] Zebulum, R.S., Stoica, A., and Keymeulen, D., A flexible model of a CMOS field programmable transistor array targeted for hardware evolution, ICES2000, Springer-Verlag, 180, 274.

[2] Gaudet, V., and Gulak, G., CMOS Implementation of a Current Conveyor-Based Field-Programmable Analog Array, 31st ASILOMAR Conference on Signals, Systems, and Computers, Pacific Grove, CA, November 1997.

[3] Hamilton, A., Papathanasiou, K., Tamplin, M. R., and Brandtner, T., Palmo, Field programmable analog and mixed-signal VLSI for evolvable hardware, in *Proceedings of the Second International Conference on Evolvable Systems: From Biology to Hardware (ICES98)*, Lausanne, Switzerland.

[4] Flockton, S. and Sheehan, K., Intrinsic circuit evolution using programmable analog arrays, in *Proc. of the Sec. International Conference on Evolvable Systems*, Sipper, M., Mange, D., and Pérez-Uribe, A., Eds., vol. 1478, LNCS, Springer-Verlag, 1998, 144.

APPENDIX A : MOS TRANSISTORS

We summarize here the basic equations that describe the operation of MOS transistors. The main objective here is to characterize the subthreshold operating region explored by evolution in the OpAmps of Chapter 4. We assume the reader to be familiar with the common terminology used to specify MOS transistors. Particularly we refer here to the equations of the nMOS transistors. The analysis for the pMOS transistor is analogous. More details can be found in Johns and Martin (1997).

First of all, we define the transistor threshold voltage V_{TN} : whenever gate to source voltage V_{GS} is larger than V_{TN}, conduction between the drain and the source will occur. The effective gate-source voltage, V_{eff}, is defined by:

$$V_{eff}, = V_{GS} - V_{TN} \tag{A.1}$$

which is proportional to the charge density Q_n that is given by:

$$Q_n = C_{ox} \cdot V_{eff}, \tag{A.2}$$

C_{ox} is the gate capacitance per unit of area, and it is given by:

$$C_{ox} = K_{ox} \cdot \varepsilon_o / t_{ox} \tag{A.3}$$

where ε_o is the permittivity of free space, K_{ox} is the relative permittivity of SiO_2, and t_{ox} is the thickness of the oxide gate. To obtain the total gate capacitance, we multiply C_{ox} by the effective area, WL (W is the width and L is the length of the channel).The total charge will then be given by:

$$Q_{Tn} = W\,L\,C_{ox} \cdot V_{eff} \tag{A.4}$$

On the other hand, if we apply a positive drain-source potential, the flowing current will be flowing through the transistor channel:

$$I_D = \mu_n Q_n (W/L) V_{DS} \tag{A.5}$$

Substituting (A.2) in (A.5):

$$I_D = \mu_n C_{ox} \ (V_{GS} - V_{TN})(W/L)V_{DS} \tag{A.6}$$

We can devise two distinct operating regions according to the value of V_{DS}. If V_{DS} is smaller than the so-called saturation region, $V_{DS\text{-}sat}$, than the current I_D is given by:

$$I_D = \mu_n C_{ox} (W/L) [(V_{GS} - V_{TN})V_{DS} - V_{DS}^2/2] \tag{A.7}$$

This is called triode or linear region. When V_{DS} is larger than V_{DS-sat}, then the current I_D is given by:

$$I_D = (\mu_n C_{ox} / 2)(W/L) [(V_{GS} - V_{TN})^2] \quad (A.8)$$

This is called saturation or active region. The saturation voltage V_{DS-sat} is given by:

$$V_{DS-sat} = V_{GS} - V_{TN} \quad (A.9)$$

being equal to the effective voltage.

The transconductance is an important parameter that characterizes the behavior of a MOS transistor, and it is given by:

$$g_m = d\, I_d / dV_{gs}$$

These equations assume that V_{eff} is greater or equal than around 100mV, a region of operation denominated strong inversion. In case V_{eff} is smaller than 100mV, the device operates in a weak inversion region. In this particular region, the behavior of the MOS transistor will be similar to a bipolar transistor, in the sense that its transconductance will approach the one of a bipolar device. In the subthreshold region the device current obeys an exponential relationship:

$$I_D = I_{D0}(W/L)e^{(qVgs/nKT)}$$

where *n* is approximately 1.5. The main advantage of the weak inversion operation is its smaller power consumption. However, the size of the device is usually increased for the device to operate in this region (large W/L). In addition, the decrease in power is accompanied by a reduction in the speed of the device. Whereas in the weak inversion region the value of g_m is around $20 \times I_D$, it is only $10 \times I_D$ in the strong inversion operating region. A bipolar transistor has a transconductance equal to about 40 times its collector current.

MOS transistors operating in weak inversion are usually employed at the input stage of low power OpAmps, as a way to explore the low DC current available and achieve a reasonable gain in the first stage of the amplifier.

The conclusion from these equations is that the behavior of the MOS device is largely influenced by its size, W/L, which gives a degree of freedom for the evolutionary optimization of circuits based upon these devices.

REFERENCES

[1] Johns, D.A. and Martin, K., *Analog Integrated Circuit Design*, John Wiley & Sons, New York, 1997.

APPENDIX B : SWITCHED CAPACITOR (SC) CIRCUITS

The operation of switched capacitor circuits relies on the *principle of resistor equivalence.*[1] This principle is illustrated in Figure B.1:

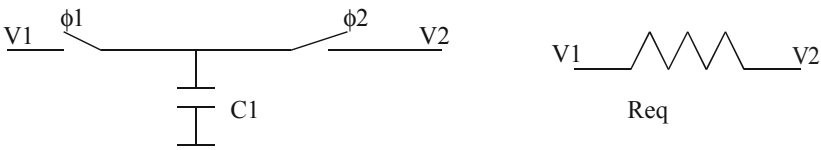


Figure B.1 Resistor equivalence principle of a switched capacitor.

We can see, in the left side of Figure B.1, a capacitor C1 between two switches controlled by the signals ϕ1 and ϕ2. These switches are implemented in practice through MOS transistors and the signals f1 and f2 are non-overlapping clocks. The charge, over one clock period, transferred to the capacitor C1 is given by:

$$Q = C1(V1 - V2) \tag{B.1}$$

The average current asociated to this charge (Coulombs per second) is:

$$I_{avg} = C1(V1 - V2)/T \tag{B.2}$$

where T is the clock period. The equivalent resistor R_{eq} in the right side of Figure B1 is then given by:

$$R_{eq} = (V1 - V2)/I_{avg} = T/C1 \tag{B.3}$$

Therefore, we can control the resistance value by increasing or decreasing the frequency of the clock signal.The distiguishing feature of SC circuits is, therefore, the use of the set capacitor + switches instead of resistors. Compared to conventional circuits, SC circuits are advantageous in terms of linearity, dynamic range, and precision. The latter is due to the fact that most of the parameters of analog circuits implemented in SC will depend on a ratio of capacitor values, which can be accurately set in most VLSI technologies.

As an example, Figure B.2 shows a conventional oscillator based on switched capacitors.

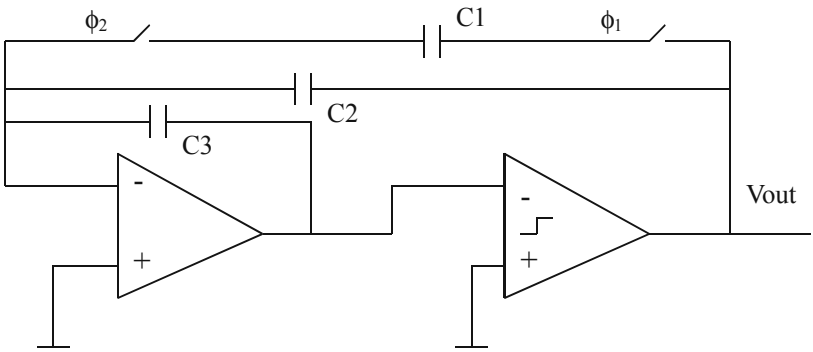


Figure B.2 SC oscillator (extracted from Johns and Martin,[1] 1997).

In this figure, the capacitor C1 replaces a resistor that would appear in a more conventionl implementation. V_{out} is the circuit output.

However, which advantages do SC technology confer to ? As it has been shown in Chapter 7, Motorola has manufactured a Field Programmable Analog Array based on SC technology, and evolutionary experiments have already been carried on these devices. The potential advantage of this approach is allowing an extra degree of freedom for GAs to evolve the values of the non-overlapping signals ϕ_1 and ϕ_2. This is a more flexible way to evolve resistance values in silicon. In addition, the area of SC circuits is not as mature as other areas of analog electronics, which leaves more room for innovative design. However, one drawback is the difficulty in getting accurate simulation of these circuits, which turns intrinsic evolution into the most suitable approach for this class of design.

REFERENCES

[1] Johns, D. A. and Martin, K., *Analog Integrated Circuit Design*., John Wiley & Sons, New York, 1997.

APPENDIX C : Gm-C FILTERS

The switched-capacitor approach for implementing integrated filters has been briefly described previously in this book. It has been explained that this approach is advantageous in terms of the filter design accuracy. Nevertheless, the disadvantage of this approach is the fact that it requires sampling in the time domain.[1] In the case of high frequency applications, a very high sampling clock rate must be achieved, which can be impractical in some situations. This appendix introduces then a second approach for realizing analog integrated filters, called *Continuous-Time Filtering*.

Transconductors are the main building blocks of continuous time filters using Gm-C technology. The transconductor produces a current output that is linearly related to an input voltage. The basic equation of the transconductor is the following one:

$$i_o = Gmv_i$$

where i_o is the output current, v_i is the input voltage, and Gm is the transconductance of the cell. This output current is applied to the integrating capacitor C1, resulting in a voltage Vo across the capacitor[1] given by:

$$V_o = I_o/sC1 = GmV_i/sC1$$

A Gm-C integrator is described in Figure C.1:

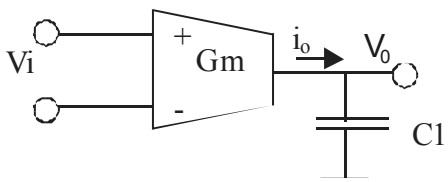


Figure C.1 Gm-C integrator (extracted from Johns and Martin,[1] 1997).

How can this structure be realized? One could think at first of an OpAmp, but, in contrast to OpAmps, Gm-C integrators present, ideally, infinite values for both the input and output impedances. This is more similar to the OTAs studied in Chapter 4, but OTA does not present the linearly input-output relationships of tranconductors. One approach to realize transconductors is through bipolar technology: a differential pair of bipolar transistors is linearized through resistor degeneration. The interested reader can consult other references in analog electronic design.

As described in Chapter 8, evolutionary design can also benefit from this technology. In this particular case, the idea is to *optimize* a continuous filter design

by adjusting the transconductance values. This procedure has already been applied for the optimization of intermediate frequency filters for mobile phones. The Gm value of the transconductor is controlled by changing its biasing current. In the application described in Chapter 8, only small adjustments of the Gm values are performed, in order to optimize the filter frequency response.

REFERENCES

[1] Johns, D.A. and Martin, K., *Analog Integrated Circuit Design*, John Wiley & Sons, New York, 1997.

APPENDIX D : AN INTRODUCTION TO NANO ELECTRONICS

D.1 INTRODUCTION

In order to continue the trend in hardware development, a new "post-lithographic" technology will be necessary: the nanotechnology, which takes into account the effects of quantum mechanics. This new technology will allow any atom to be put in the right place, and thus, the construction, at low cost, of any molecular structure consistent with the laws of physics and chemistry.

Quantum Mechanics Effects

When the size of metallic or semiconductor islands is in the nanometer scale, the total capacitance becomes very small and the charging energy is larger than the thermal energy. Therefore, the addition or subtraction of a single electron from the island, or a quantum dot, causes a change in the free energy that becomes significant.

In these circumstances, new phenomena appear, such as the Coulomb blockade, which is a suppression of current flow at low bias, and Coulomb oscillations, a time or space correlated transfer of electrons through tunnel junctions. These new quantum-effects allow the control of the movement and position of single electrons.

However, unwanted effects also arise, such as co-tunneling, which is a simultaneous tunneling of two or more electrons in different tunnel junctions. The increased sensitivity to uncontrollable impurities that are dispersed throughout the material, disturbing the charge distribution and hence the Coulomb blockade, also turns more difficult the controlled transfer of single electrons.

The Single-Electron Transistor

Several types of quantum devices are under development.[1] One of them is the single-electron transistor which is under development by research groups worldwide. Though quantum transistors are a novelty today, they will be needed once the classical field-effect transistor (FET) can be made no smaller. In quantum transistors, electrons skip on and off quantum dots or tunnel through barriers thought impenetrable in the world of classical physics.

Due to capacitance and thermal fluctuation limitations, the island size of the Single-Electron Transistor developed by Matsumoto at Stanford[2] could be no larger than ~10 nm. This size is out of the range of present conventional microfabrication processes.

Using an artificial pattern formation method based on the scanning tunneling microscope (STM), they succeeded in fabricating a SET. The SET operates at room temperature, showing a clear Coulomb staircase with a ~150 mV period at 300 K.

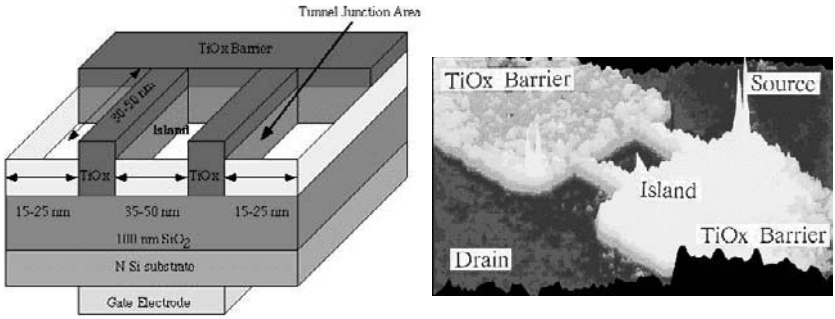


Figure D.1 Schematic of a single electron transistor and its AFM image (extracted from[2]).

Figure D.1 illustrates the SET made by the STM nano-oxidation process. At both ends of the 3 nm thick Ti layer the source and drain ohmic contacts were formed, and on the backside of the n-Si substrate, the gate ohmic contact was formed. At the center region of the Ti layer, the island region was formed, surrounded by two parallel, narrow TiOx lines that serve as tunneling junctions for the SET, and two large TiOx barrier regions. Figure D.1 also shows an atomic force microscopy (AFM) image of the island region of a fabricated SET.

Due to this small tunneling junction area, the tunnel capacitance becomes as small as 10^{-19}F, which allows the SET to be operated at room temperature.

D.2 A PLATFORM FOR THE EXTRINSIC EVOLUTION OF NANOMETRIC CIRCUITS

With the appearance of nanotechnology, the complexity of synthesis problems will increase in a very large scale. As Genetic Algorithms are able to deal with complex optimization problems, GAs may turn into a key technology in the following years. Some work has already been done in this field, such as the optimization of single-electron NOR gates through Genetic Algorithms [3, 4], although these are still restricted to simulation. As an example, we will describe the creation of a platform for the extrinsic evolution of single electron circuits in nanometric scale. In order to create the platform, the SIMON1.1[5] simulator and the genetic algorithms tool GNOCOP3[6] were used.

The Simulator of Nanostructures

SIMON (Simulator of Nanostructures) is a single electron circuit simulator. The simulated circuits consist of webs of tunnel junctions, capacitors, constant voltage sources, piece-wise linearly time dependent voltage sources, and voltage-controlled voltage sources. The electrons move through the tunnels with certain probabilities and receive energy from the voltage sources and the environment.

The simulator is based on a Monte Carlo (MC) method where the free energy before and after any particular tunnel event determines the tunnel rate. Among all

possible events one is chosen, according to the computed probability distribution, as the winner, which is taken as the actual happening event. By simulating many tunnel events the macroscopic device characteristics are obtained.

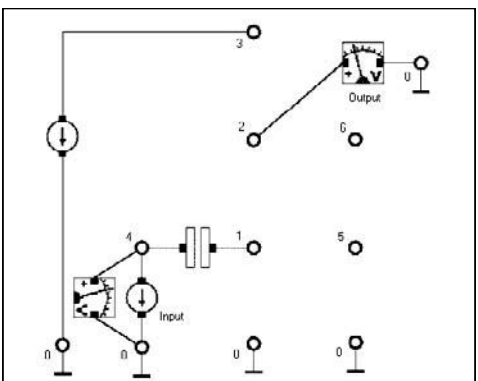
One may choose between transient and stationary simulation modes and decide upon the order of co-tunneling which should be included in the simulation. Other parameters are also set, such as the temperature of the environment and the execution step.[3]

The Evolution of an Inverter Circuit

In order to evolve circuits, the interface between GNOCOP3 and SIMON was implemented so that the communication could occur through text files. An inverter circuit was evolved in order to evaluate the platform's performance.

Each position or *locus* of a chromosome can be represented in many ways, usually through bits or integers. For the evolution of the inverter circuit, the chromosome had 8 genes to be evolved. Each gene represented the type of component connected with each pair of nodes and the values of the resistances and capacitances associated with the component. Therefore, the gene had 3 positions: the first position represented the component nature, which could be a capacitor (value 1), a tunnel junction (value 2), a wire (value 3), or an open circuit (value 0); the second position represented the value of the capacitance of the component, which was taken into account for capacitors or tunnel junctions; finally, the last position represented the value of the resistance of the component, which was taken into account only for tunnel junctions. Figure D.2 shows the fixed parts of the circuit and the position of the component represented by each gene.

The values of the reference voltage source and the positive and negative references for the input signals, as well as some other components of the circuit, were fixed and could not be evolved. All simulations were made at 0 Kelvin and the execution step was of 10^{-2} s; therefore, 100 points were calculated each second.



Gene=[component nature, capacitance value, resistance value]

Gene 1: nodes 0 and 1	Gene 5: nodes 5 and 6
Gene 2: nodes 1 and 2	Gene 6: nodes 6 and 3
Gene 3: nodes 2 and 3	Gene 7: nodes 6 and 2
Gene 4: nodes 0 and 2	Gene 8: nodes 5 and 1

Range of values of the capacitance: from 10^{-16} F to $5x10^{-15}$ F
Range of values of the resistance: from $5x10^{4}\Omega$ to $1.5x10^{5}\Omega$

Figure D.2 The representation of the circuit to be evolved.

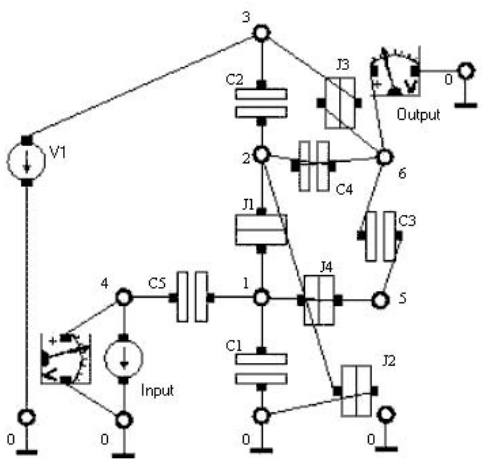
Results

The final circuit was obtained after approximately 3 hours of execution in a Pentium II 300MHz. Figure D.3 shows the best chromosome found, without the values that are not taken into account.

```
[ (1, 1.265e-15, – ), (2, 1.547e-15, 9.505e+04),
  (1, 4.969e-15, – ), (2, 1.641e-15, 1.032e+05),
  (1, 3.752e-15, – ), (2, 1.784e-15, 1.292e+05),
  (1, 3.836e-15, – ), (2, 4.093e-15, 1.155e+05) ]
```

Figure D.3 The best chromosome found.

The final circuit, along with the evolved and fixed values, can be seen on Figure D.4.



C1: C=1.265e-15 F
C2: C=4.969e-15 F
C3: C=3.752e-15 F
C4: C=3.836e-15 F
J1: C=1.547e-15 F R=9.505e+04 Ω
J2: C=1.641e-15 F R=1.032e+05 Ω
J3: C=1.784e-15 F R=1.292e+05 Ω
J4: C=4.093e-15 F R=1.155e+05 Ω
Fixed:
C5: C=5.000e-16 F
V1: V=1.500e-05 V

Figure D.4 The evolved circuit.

The graph of input and output tensions of the inverter is presented on Figure D.5.

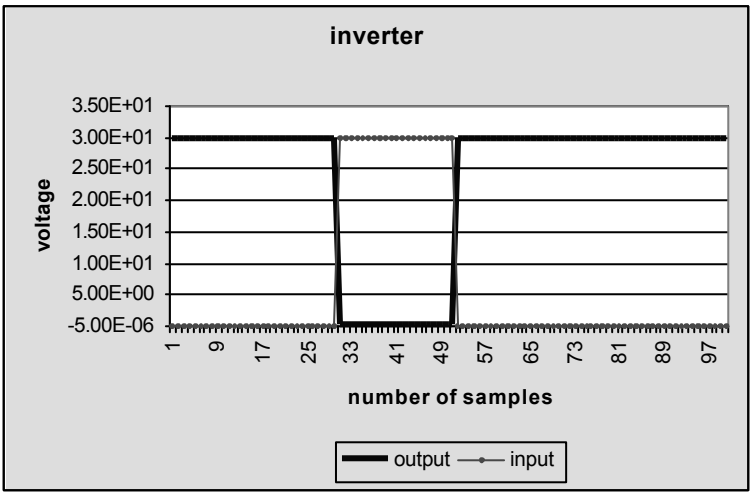


Figure D.5 Results of the evolved circuit.

The results show the good performance of the platform. Although simple, the evolution of an inverter circuit is the first step for the evolution of more complex circuits. In order to make the simulation and evolution more realistic, the effects of co-tunneling can be taken into account, as well as the temperature parameter can be fixed at higher values. The number of components to be evolved could also increase, in order to demonstrate better the exploitation of electronics design through artificial evolution. Circuits that are beyond the scope of conventional design methodologies are evolved through artificial evolution in the "macro" world, and this characteristic is shown to exist also in the single-electronics.

REFERENCES

[1] *Quantum Transistors: Toward Nanoeletronics*, IEEE Spectrum, 2000.

[2] Matsumoto, K., *Single Electron Transistors*, http://luciano.stanford.edu/~shimbo/set.html.

[3] Thompson, A. and Wasshuber, C., Evolutionary design of single electron systems, *The Second NASA / DoD Workshop on Evolvable Hardware*.

[4] Chen, R., Korotkov, A., and Likharev, K., Single-electron transistor logic, *Appl. Phys. Lett*., 1996.

[5] Wasshhuber, C., Kosina, H., and Selberherr, S., SIMON, a simulator for single-electron tunnel devices and circuits, *IEEE Transactions on Computer-Aided Design*, Sept., 1997.

[6] Michalewicz, L., *Genetic Algorithms + Data Structures = Evolution Programs*, Spring-Verlag, New York, 1994.