

On The Extension of ECGA for Different Variable Types: Integers and Real Numbers

Ping-Chu Hung

NCLab Report No. NCL-TR-2007010

June 2007

Natural Computing Laboratory (NCLab)
Department of Computer Science
National Chiao Tung University
329 Engineering Building C
1001 Ta Hsueh Road
HsinChu City 300, TAIWAN
<http://nclab.tw/>

國立交通大學

多媒體工程研究所

碩 士 論 文

擴充 ECGA 於不同的資料型別：
整數與實數



On The Extension of ECGA for Different Variable Types:
Integers and Real Numbers

研 究 生：洪秉竹

指導教授：陳穎平 教授

中 華 民 國 九 十 六 年 六 月



擴充 ECGA 於不同的資料型別：整數與實數
On The Extension of ECGA for Different Variable Types:
Integers and Real Numbers

研 究 生：洪秉竹

Student：Ping-Chu Hung

指導教授：陳穎平

Advisor：Ying-Ping Chen

國立交通大學
多媒體工程研究所
碩士論文



Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

June 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年六月



摘要

延伸式精簡基因演算法(ECGA)是一種能解決二進位難題的演算法。因為具有偵測建構區塊(building blocks)的能力，ECGA 可靠而且精確。但當我們直接應用 ECGA 於整數問題時，仍會遇到某些困難。在本研究中，我們提出一種延伸 ECGA 的新演算法，稱為整數型延伸式精簡基因演算法(iECGA)。iECGA 使用修改過的機率模型並繼承了偵測建構區塊的能力。iECGA 特別設計來處理整數問題，並能避免 ECGA 遇到的困難。

為了解決固態元件中的特性量定(characteristic determination)問題，我們也發展了一種新的最佳化架構，包含了 ECGA 及一種離散化技巧稱為隨選分割(SoD)。因為特性量定問題中的變數因為物理性質，所以幾乎都是實數，ECGA 只能處理離散型的問題，因此需要一種機制轉換資料型態。所以在本研究中，我們將提出的架構應用在三個研究個案上，並展示這個演化計算領域中的方法，不止提供了高品質的最佳化結果，也有處理不同問題的彈性。

關鍵字：基因演算法、延伸式精簡基因演算法、隨選分割、建構區塊、特性量定

Abstract

Extended compact genetic algorithm (ECGA) is an algorithm that can solve hard problems in the binary domain. ECGA is reliable and accurate because of the capability of detecting building blocks, but certain difficulties are encountered when we directly apply ECGA to problems in the integer domain. In this paper, we propose a new algorithm that extends ECGA, called integer extended compact genetic algorithm (iECGA). iECGA uses a modified probability model and inherits the capability of detecting building blocks from ECGA. iECGA is specifically designed for problems in the integer domain and can avoid the difficulties that ECGA encounters.

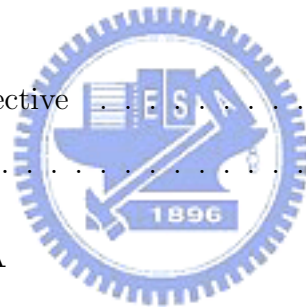
We also develop a new optimization framework that consists of the extended compact genetic algorithm (ECGA) and split-on-demand (SoD), an adaptive discretization technique, to tackle the characteristic determination problem for solid state devices. As most decision variables of characteristic determination problems are real numbers due to the modeling of physical phenomena, and ECGA is designed for handling discrete-type problems, a specific mechanism to transform the variable types of the two ends is in order. Therefore, in this study, we employ the proposed framework on three study cases to demonstrate that the technique proposed in the domain of evolutionary computation can provide not only the high quality optimization results but also the flexibility to handle problems of different formulations.

keywords:

Genetic algorithms, extended compact genetic algorithms, iECGA, split-on-demand, building blocks, characteristic determination

Contents

Abstract	i
Table of Contents	iii
List of Figures	v
List of Tables	vii
1 Introduction	1
1.1 Motivation and Objective	2
1.2 Road Map	2
2 Brief Review of ECGA	4
2.1 Genetic Algorithms and Linkage Problem	4
2.2 Linkage Learning and Probability Model	5
2.3 Extended Compact Genetic Algorithm	7
2.4 ECGA on Trap Problems	8
2.5 Problems in Integer Domain	9
3 Extend ECGA to Integer Domain	11
3.1 Marginal Product Model	11
3.2 MDL Model	12
4 Performances of iECGA	13
4.1 Test Functions	13
4.2 Experiments and Parameters	15
4.3 Experimental Results	16



4.4	Discussion	17
5	Extend ECGA to Real-valued Domain	25
5.1	Split on Demand for Discretization	25
5.2	ECGA with SoD	27
6	Apply rECGA on Characteristic Determination Problem	28
6.1	Conventional TFT	28
6.2	TFT under High Gate Bias	32
6.3	Frequency Response	34
7	Conclusions	42
7.1	Summary	42
7.2	Future Work	43
7.3	Main Conclusions	43



List of Figures

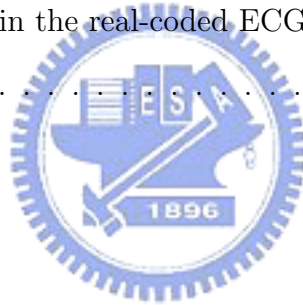
4.1	The suboptimum of $f_2(x_1x_2)$ is at $(0,0)$, but the optimum of $f_2(x_1x_2)$ is at $(7,7)$	15
4.2	The average(a) and best(b) fitness of three algorithms in f_1 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.	19
4.3	The average(a) and best(b) fitness of three algorithms in f_2 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.	20
4.4	The average(a) and best(b) fitness of three algorithms in f_3 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.	21
4.5	The average(a) and best(b) fitness of three algorithms in f_4 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.	22
4.6	The convergence speed of f_1 and f_2	23
4.7	The convergence speed of f_3 and f_4	24
5.1	Populations and possible split positions (vertical lines). The numbers close to the positions are the order in which the positions are decided.	26
6.1	The structure and the high-conducting channel formed for the conventional poly-Si TFT.	29
6.2	Experimental data and the match results for study case I.	30
6.3	TFT under high gate bias.	33

6.4	Experimental data and the match results for study case II. ELA, SSL, FLA, and SPC are four different kinds of TFTs.	37
6.5	Experimental data and the match results for study case II. ELA, SSL, FLA, and SPC are four different kinds of TFTs.	38
6.6	Structure of gate/SiO ₂ /poly-Si and its equivalent circuit.	39
6.7	Frequency response for TFTs.	40
6.8	Match results for the frequency response.	41



List of Tables

2.1	An example MPM for four genes	6
2.2	The output of ECGA	8
3.1	An MPM example in ECGA	12
3.2	An MPM example in iECGA	12
4.1	The cardinalities (d) and the length of building blocks (BB) of f_2 , f_3 , and f_4	17
6.1	Parameters adopted in the real-coded ECGA for handling the three study cases.	31



Chapter 1

Introduction

Genetic Algorithms (GAs), proposed by John Holland [1], are powerful search techniques based on principles of evolution. The solutions to our problems are represented as binary strings, which are called “chromosomes”. Different evolutionary operators, such as mutation and crossover, are performed on the chromosomes to generate “offsprings”. In each generation, the offsprings are evaluated and selected as new parents. During the process of evolution, the fitness of solutions becomes higher, and the best solution are chosen as the final answer. Based on the concept of GA, many variations of GA are proposed, and the original GA is called simple GA.

The traditional genetic operators such as one-point crossover and bitwise mutation cannot appropriately handle the problems that require the utilization of linkage information [2, 3]. Many problems in practical cannot be solved completely without the ability of linkage learning, thus the concepts of genetic linkage and building blocks are the essential components of GAs [4, 5]. As a consequence, the processing of genetic linkage, such as detection and utilization, has attracted much attention in the field of evolutionary and genetic computation.

The most popular way to gather linkage information from genes is to collect and process the global, population-wise statistics. These variations of GAs based on statistics are called estimation of distribution algorithms (EDAs) [6, 7]. Some EDAs assume all genes or variables are independent, such as the population based incremental learning (PBIL) [8], the univariate marginal distribution algorithm (UMDA) [9], and the compact genetic algorithm (cGA) [10]. The others compute the dependencies and/or relations

between genes, such as the Bayesian optimization algorithm (BOA) [11], and the extended compact genetic algorithm (ECGA) [12].

1.1 Motivation and Objective

The idea of ECGA is to solve hard problems by learning genetic linkage on the fly. ECGA employs the marginal product model (MPM) to represent the joint probability distribution of genes or variables and adopts the minimum description length (MDL) as the criterion to determine how good the learned joint distribution is. Harik's numerical experiments indicate that ECGA has better performance than a simple GA does when solving hard problems. When tackling binary optimization problems, ECGA is reported quick, reliable, and accurate. However, if ECGA is applied on integer optimization problems, can it perform as well as it can on binary ones? Moreover, if ECGA is applied on real-valued problems, how to encode real numbers such that the advantages of ECGA can be kept.

The easiest way to solve non-binary problems by ECGA is changing its representation—encoding integers and real numbers as binary strings. When the cardinality of integers is power of two, encoding does not complicate the problem. If the cardinality is not power of two, encoding may cause perturbation of linkage information. In this paper, we reveal the disadvantages when ECGA solves integer optimization problems. We also propose a new algorithm called the integer extended compact genetic algorithm (iECGA) that can efficiently solve hard integer problems.

When encountering real-valued problems, real extended compact genetic algorithm (rECGA) [13], which employs a novel representation schema Split-on-Demand, is a suitable tool for solving problems. In this paper, we will apply rECGA on a real-world problem.

1.2 Road Map

The remainder of this book is structured in what follows. Chapter 2 presents a brief introduction of extended compact genetic algorithms. The Chapter also presents the probability models used by ECGA, which are called marginal product models (MPM).

Chapter 3 provides the discussion of problems in representing integers. We make the genes be represented as integers and modify probability models to match up integers. Then, Chapter 4 shows the experimental results of iECGA on “GA hard” problems of different scales. All contents about integer ECGA are represented.

Chapter 5 describes the representation of real numbers in rECGA [13], which is called Split-on-Demand. The Chapter also describes the procedure of rECGA. Chapter ?? describes the problems of characteristic determination in solid state devices. And the Chapter provides the experimental result of rECGA on characteristic determination problems. Finally, Chapter 7 concludes this paper by summarizing its contents, discussing important results, showing possible future works, and offering out conclusions.



Chapter 2

Brief Review of ECGA

This chapter provides a brief review of genetic algorithms and introduces the term *genetic linkage problem*. We also explain why linkage learning is one of the most important topics in the field of genetic and evolutionary algorithm. After introducing GA and linkage learning, we show the idea of compact genetic algorithm (cGA) [10] that the population of GAs can be viewed as a probability model. Then We review extended compact genetic algorithm (ECGA) [12, 14] that based on the concept of cGA.

2.1 Genetic Algorithms and Linkage Problem

For all creatures live in Nature, survival of the fittest happens everyday. The environment gives all creatures a contest, and the penalty for losers is losing their lives. The contest is so-called “selection”. During the process of natural selection, useless individuals are eliminated, and strong or smart individuals can keep survive. Genetic algorithms are proposed based on the concept of natural selection. In most genetic algorithms, the life cycle contains three steps: selection, breeding, and evaluation. In selection step, parents are selected from population. Then some operators like crossover and mutation are used on parents to produce offsprings. The new offspring are evaluated and become new population.

Since genetic algorithms are proposed, the importance of building blocks (BBs) and their role in the working of GAs have long been recognized [1, 5]. Most problems are composed of smaller sub-problems. That is, the solutions to problems can be decomposed into several parts, which are called “building blocks”. During the process of simple GA,

BBs are often destructed by one-point or uniform crossover. If we can identify BBs correctly and mixed them without destruction, good solutions will emerge more quickly. The problem of identifying BBs are called “linkage problem”.

The most popular way to solve linkage problem is to collect and process the global, population-wise statistics. These variations of GAs based on statistics are called estimation of distribution algorithms (EDAs) [6, 7]. The extended compact genetic algorithm (ECGA) [12], which is based on the compact genetic algorithm (cGA) [10], is a robust and efficient GA that has linkage learning ability. In the next section, we will introduce how ECGA learns linkage from probability models.

2.2 Linkage Learning and Probability Model

There are two important assertions behind the concept of ECGA. Firstly, learning a “good” probability distribution is equivalent to learning genetic linkage. Secondly, the “goodness” of a probability distribution is based on how much computational resource, mainly the space, the computer system needs to store the population and the distribution. The first assertion makes it rational to learning linkage from probability models, and the second assertion shows one way to define the quality of a probability model.

Compact genetic algorithm (cGA) models a binary GA population as a vector of probability distribution. Assume an individual is a n -bit binary string $b_1b_2\dots b_n$, and $P[i]$ is the probability that $b_i = 1$. For example, if 70 percents of individuals have one on the first bit, $P[1] = 0.7$. Since the population can be modeled as distribution, the crossover operator can be modeled as operation on distribution. Thus, finding the optimal solution in cGA is equivalent to finding the optimal probability distribution. Because the probability of each gene is independent, cGA does not have the ability to maintain linkage information.

ECGA extends the probability model in cGA from a probability vector to the marginal product model (MPM). MPMs are similar to the models employed by cGA and PBIL, except that they can represent the joint probability distribution over more than one gene at a time. As an example, a simple MPM is shown in Table 2.1. MPM divides the genes or

group [0 3]		group [1]		group [2]	
allele	prob.	allele	prob.	allele	prob.
00	0.1	0	0.5	0	0.6
01	0.3	1	0.5	1	0.4
10	0.2				
11	0.4				

Table 2.1: An example MPM for four genes

variables into several groups. In Table 2.1, four genes are divided into three groups [gene 0, gene 3], [gene 2], and [gene 1]. For each group, we count the occurrence of different patterns in the whole population and store it in the table. We choose the MPM for two reasons: 1) they make the exposition simpler; and 2) the structure of the model can be directly translated into a linkage map.

The object of ECGA is to find “good” distributions. How do we define the criterion to judge the goodness of different probability distributions? The idea is to adopt the concept of *Occam’s Razor* long recognized in the domain of machine learning [15]:

By reliance on Occam’s Razor, good distributions are those under which the representation of the distribution using the current encoding, along with the representation of the population compressed under that distribution, is minimal.

Thus we know that good distribution has two criteria: small model representation and small population representation. One way to realize this concept is the minimum description length (MDL) principle [16]. Following the definition, we can use the MDL model on MPMs and define the model complexity and the compressed population complexity of a probability distribution as

$$\text{Model Complexity} = \log_2 N \sum_{i=1}^m 2^{s_i} \quad (2.1)$$

and

$$\text{Compressed Population Complexity} = N \sum_{i=1}^m \sum_p -p \log_2 p, \quad (2.2)$$

where m is the number of groups, s_i is the size of i th group, p is the probability of an allele pattern in i th group, and N is the population size. The combined complexity is the summation of the model complexity and the compressed population complexity.

2.3 Extended Compact Genetic Algorithm

In the previous section, we know how to judge the goodness of a probability distribution. Now the problem is how to find a most suitable distribution for a population. We use greedy search to find the most suitable distribution.

Assume the length of binary string is L . First, we assume all genes are independent and each gene forms a separate group, that is, the MPM $[0][1] \dots [L-2][L-1]$ is the starting model for the building process. Then, we try to merge each pair of groups into a new distribution. As a result, $[0, 1] \dots [L-1]$, $[0][1, 2] \dots [L-1]$, \dots , and $[0][1] \dots [L-2, L-1]$ are produced. For every produced MPM, the combined complexity is calculated, and we compare all complexities as well as original MPM. If a produced MPM $[0, 1] \dots [L-1]$ has smallest complexity, group $[0]$ and group $[1]$ are combined as a new group $[0,1]$. The combination process continues until it is impossible for any improvement on complexity. After the process stops, we have a MPM representing the linkage between genes and can use the configuration to perform crossover.

The procedure of ECGA is similar to that of a simple GA: initialization, evaluation, parent selection, and crossover. The difference between ECGA and a simple GA is that ECGA models the probability distribution of the parents. Then a greedy MPM search as mentioned before is used to find the linkage information. After all, BB-wise crossover is performed by utilizing the linkage information. Since the BBs will not be destroyed by crossover operator, the speed of evolution will be faster than simple GA. Algorithm 1 shows the procedure of ECGA.

Algorithm 1 The procedure of ECGA

```

Generate individuals at random
Generation  $\leftarrow 1$ 
while Generation < maxGen do
    Calculate fitness values of individuals
    Perform tournament selection
    Use MPM to build a joint probability distribution
    Use the generated MPM to perform crossover
    Generation  $\leftarrow$  Generation + 1
end while
Report the result

```

Generation	Marginal Product Model
1	$\begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 32 & 33 & 34 & 35 \end{bmatrix} \begin{bmatrix} 39 \end{bmatrix} \begin{bmatrix} 16 & 18 & 20 & 22 \end{bmatrix} \\ \begin{bmatrix} 21 & 23 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \begin{bmatrix} 12 & 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} 36 & 37 & 38 \end{bmatrix} \\ \begin{bmatrix} 19 & 27 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \end{bmatrix} \begin{bmatrix} 17 & 28 & 29 & 30 & 31 \end{bmatrix} \begin{bmatrix} 24 & 25 & 26 \end{bmatrix} $
2	$\begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 32 & 33 & 34 & 35 \end{bmatrix} \begin{bmatrix} \mathbf{36} & \mathbf{37} & \mathbf{38} & \mathbf{39} \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \\ \begin{bmatrix} \mathbf{24} & \mathbf{25} & \mathbf{26} & \mathbf{27} \end{bmatrix} \begin{bmatrix} 12 & 13 & 14 & 15 \end{bmatrix} \begin{bmatrix} 28 & 29 & 30 & 31 \end{bmatrix} \\ \begin{bmatrix} 16 & 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 8 & 9 & 10 & 11 \end{bmatrix} \begin{bmatrix} \mathbf{20} & \mathbf{21} & \mathbf{22} & \mathbf{23} \end{bmatrix} $
3	$\begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 28 & 29 & 30 & 31 \end{bmatrix} \begin{bmatrix} 20 & 21 & 22 & 23 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \\ \begin{bmatrix} 36 & 37 & 38 & 39 \end{bmatrix} \begin{bmatrix} 32 & 33 & 34 & 35 \end{bmatrix} \begin{bmatrix} 12 & 13 & 14 & 15 \end{bmatrix} \\ \begin{bmatrix} 8 & 9 & 10 & 11 \end{bmatrix} \begin{bmatrix} 16 & 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 24 & 25 & 26 & 27 \end{bmatrix} $
4	$\begin{bmatrix} 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 28 & 29 & 30 & 31 \end{bmatrix} \begin{bmatrix} 24 & 25 & 26 & 27 \end{bmatrix} \begin{bmatrix} 4 & 5 & 6 & 7 \end{bmatrix} \\ \begin{bmatrix} 36 & 37 & 38 & 39 \end{bmatrix} \begin{bmatrix} 32 & 33 & 34 & 35 \end{bmatrix} \begin{bmatrix} 12 & 13 & 14 & 15 \end{bmatrix} \\ \begin{bmatrix} 8 & 9 & 10 & 11 \end{bmatrix} \begin{bmatrix} 16 & 17 & 18 & 19 \end{bmatrix} \begin{bmatrix} 20 & 21 & 22 & 23 \end{bmatrix} $

Table 2.2: The output of ECGA

2.4 ECGA on Trap Problems

In this section, we will show the performance of ECGA to solve a trap function [17, 18] as an example. Trap functions are considered as fundamental components of *GA-hard* problems and are usually chosen to test the functionality of learning genetic linkage [12, 3].

A 4-bit trap function can be defined as $g_1 : \mathbb{Z}_2^4 \rightarrow \mathbb{Z}$ by

$$g_1(x_1x_2x_3x_4) = \begin{cases} 5, & \text{if } x_i = 0 \text{ for all } i, \\ x_1 + x_2 + x_3 + x_4, & \text{otherwise.} \end{cases}$$

This function is called “trap” function because such functions exhibit a local optimum towards which the population converges. $g_1(1101)$ is greater than $g_1(0101)$, so we better put a one in the first bit. $g_1(0101)$ is greater than $g_1(0001)$, so we better put a one in the second bit. As a result, the population will converge at 1111, where $g_1(1111) = 4$ is the local optimum.

Several small trap functions can constitute a big function. Ten 4-bit trap function constitute a 40-bit GA-hard function. From the literature, we know that a simple GA without the capability of learning genetic linkage cannot find the optimal solution to g_1 if the chromosome encoding is not appropriate [19]. Only *competent* GAs [4] that can regard related four bits as one building block are able to find the optimal solution. That is why we take trap functions to verify the ability of ECGA.

In this simple test, the population size is 1000, and tournament size is 32. The MPM

models searched in each generation is shown in Table 2.2. In the first generation, ECGA does not detect all building blocks correctly, but it soon finds all building blocks in second generation. The optimal solution “0000 ... 0000” is found in 4th generation. Detailed experiments and analysis are provided in [12, 14].

2.5 Problems in Integer Domain

When we directly apply an algorithm that solves problems in binary domain, such as ECGA, to problems in integer domain, certain difficulties will be encountered. The first difficulty is the gap between the genotype and the phenotype. That is, if the cardinality of an integer is not power of two, we have to choose a nearest number of bits to represent the integer. An integer ranging from 0 to 15 needs 4 bits to represent, but an integer ranging from 0 to 10 still need 4 bits to represent. It is easy to convert an integer to binary string, but truncation will happen in opposite converting.

Hence, to solve the representation gap, there are two general ways. One is to limit the chromosome in a given range. All genes generated out of range will be discarded. Another way is to map redundant binary strings onto the same slot. Both ways have their disadvantages. Let take integers ranging from 0 to 10 as individuals and integer value as the fitness. If $f(1010) = 10$ and $f(0111) = 7$ are chosen as parents, 6 kinds of offsprings must be discarded, and the average fitness of the remainder is only 5.6. Because most offsprings begin at 1 are discarded, the fitness of the remainder becomes lower. Let try another way where all individuals greater than 10 are mapped to modular 10, then the average fitness of offsprings of $f(1010) = 10$ and $f(0111) = 7$ is 4.75! In both ways, we try to mix two superior individuals, but the results do not preserve the quality of parents. The gap between integers and binaries cancels the advantage of GAs.

The second difficulty comes from the linkage learning ability of ECGA. In the case a problem is composed of several integer problems, the bits that belong to the same integer have linkage, and the integers that belong to the same building block also have linkage at a higher level. In order to correctly find all building blocks, ECGA needs to discover genetic linkage at two different levels. The extra computational cost may cause ECGA

inaccurate and unreliable. Moreover, the linkage ECGA finds at the bit level may not be the actual linkage at the integer level at which we are solving the optimization problem.

One simple way to overcome these difficulties is to adopt the integer representation. By using an integer vector to represent integers, there is no gap between the phenotype and the genotype, and the linkage between the bits of the same integer, which is obvious in integer optimization problems, is implicitly recognized. Therefore, an integer version of ECGA is in order.



Chapter 3

Extend ECGA to Integer Domain

In this section, we propose a modified version of ECGA, integer extended compact genetic algorithm (iECGA) [20]. All genes are represented as integers. The marginal product models and MDL criterion are modified to fit new representation.

3.1 Marginal Product Model

We define an integer as ranging from lower bound l to upper bound u . The cardinality is $d = u - l + 1$. An individual in iECGA is an integer vector, instead of a bit vector in the original ECGA. In order to compare iECGA and ECGA in a pair point, we let the cardinality be a power of two to avoid the gap between the phenotype and the genotype. We choose $l = 0$ and $u = 15$ or $u = 7$, such that the cardinality will be 16 or 8, which simplify the difficulties to represent an integer in ECGA and GA.

In ECGA, the implementation of MPM is a counting process. Let take an example that $s = [1, 3, 4]$ is a group of genes and $|s| = 3$ is the size of s . The example is shown in Table 3.1. We count the occurrences of all possible patterns in the population (* means “don’t care”), which is equivalent to the probability of the corresponding pattern. The occurrence of 0^*01^* is two, so the probability is $\frac{2}{n}$ where n is the population size.

In iECGA, we also have to count the occurrences of all possible patterns. Given the upper bound u and the lower bound l , the cardinality of the domain is $d = u - l + 1$. There are $d^{|s|}$ patterns for a group of size $|s|$. If we want to build the MPM, we have to count all $d^{|s|}$ patterns. For example, in Table 3.2, the upper bound is $u = 7$, the lower bound is $l = 0$, and the cardinality is $d = 8$. If the group of genes is $[1, 3]$, we have to

Current Population	Pattern	Count
00110	0*00*	0
01010	0*01*	2
01110	0*10*	1
01100	0*11*	2
00010	1*00*	1
10001	1*01*	0
	1*10*	0
	1*11*	0

Table 3.1: An MPM example in ECGA

Current Population	Pattern	Count
3472	0*0*	0
1624	0*1*	1
0314	0*2*	0
6715	\vdots	\vdots
4360	7*6*	1
7164	7*7*	0

Table 3.2: An MPM example in iECGA

count the occurrence of 8^2 patterns in the population.

3.2 MDL Model

Beside modification of MPMs, we also have to modify the formula of complexity. Good distribution has two criteria: small model representation and small population representation. These criteria are still applied to integer vectors. The base number in the formula of model complexity is changed from 2 to d .

$$\text{Model Complexity} = \log_2 N \sum_{i=1}^m d^{s_i} \quad (3.1)$$

And the compressed population complexity is invariant.

$$\text{Compressed Population Complexity} = N \sum_{i=1}^m \sum_p -p \log_2 p, \quad (3.2)$$

The procedure of iECGA is almost the same as ECGA, except the MPMs and the formula of complexity. The linkage information obtained in greedy MPM search is used to perform crossover. In the next section, we will show the difference between iECGA and ECGA and why integer representation is a must.

Chapter 4

Performances of iECGA

The purpose of this section is to show the difference between iECGA and ECGA. We first define some GA-hard functions as fitness functions. Then we use GA, iECGA, and ECGA to solve these functions. Finally, we will discuss the performance and properties of iECGA.

4.1 Test Functions

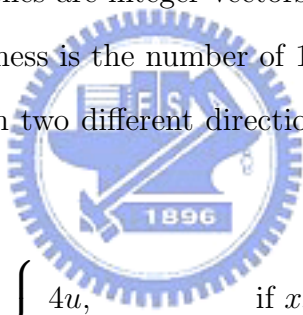
The test functions required in this study should have certain trap structure in the fitness landscape, so an algorithm cannot find the optimal solution without learning genetic linkage. A *deceptive* function is one of such a function in which the low-order schema fitness averages favor a particular local optimum, but the global optimum is located at the complement of that local optimum's position [19, 21]. To solve a deceptive function, GA must have the ability to learn linkage. Therefore, we choose deceptive functions as the basic components of our test functions.

The purpose of the experiments is to show whether iECGA outperforms ECGA in the integer domain. Here we define the following four test functions in the integer domain. Each test function is composed of several smaller deceptive functions. For example, if the input length of a deceptive function is two integers, a test function with input length 10 is composed of five deceptive functions. In the following function definitions, we assume u is the upper bound of a integer and the lower bound is 0.

$$f_1(x) = \begin{cases} 5, & \text{if } x = 0 \\ 1, & \text{if } x = 1, 2, 4, 8 \\ 2, & \text{if } x = 3, 5, 6, 9, 10, 12 \\ 3, & \text{if } x = 7, 11, 13, 14 \\ 4, & \text{if } x = 15 \end{cases}$$

The idea of f_1 is simple. It is just a deceptive one-max function. A one-max function counts the number of 1's in a binary string, and the fitness is equivalent to the number of 1's. If the string contains k 1's, the fitness is k . The deception happens when the string contains all 0's. 0000 has the highest fitness.

The input of f_1 is an integer range from 0 to 15, and then the integer is decomposed into bits to count the number of ones. In the integer level, there is no building block, so f_1 is easy for GA, in which genes are integer vectors, to solve. But if we see it at the bit level, we will find that the fitness is the number of 1's. Since the local optimum and the global optimum are located in two different directions, f_1 is a deceptive function at the bit level.



$$f_2(x_1x_2) = \begin{cases} 4u, & \text{if } x_i = u \text{ for } i = 1, 2 \\ 2u - x_1 - x_2, & \text{otherwise} \end{cases}$$

f_2 is also a deceptive function. The inputs of f_2 are two integers. In general, when x_1 and x_2 becomes larger, the fitness is smaller. But the deception happens at the largest value of x_1 and x_2 . Figure 4.1 shows the landscape of f_2 .

$$f_3(x_1x_2x_3) = \begin{cases} 6u, & \text{if } x_i = u \text{ for } i = 1, 2, 3 \\ 3u - x_1 - x_2 - x_3, & \text{otherwise} \end{cases}$$

$$f_4(x_1x_2x_3x_4) = \begin{cases} 8u, & \text{if } x_i = u \text{ for } i = 1, 2, 3, 4 \\ 4u - x_1 - x_2 - x_3 - x_4, & \text{otherwise} \end{cases}$$

f_3 and f_4 are both designed for the integer domain. The global optimum is located at (u, u, \dots, u) , and the local optimum is located at $(0, 0, \dots, 0)$. To solve these functions,

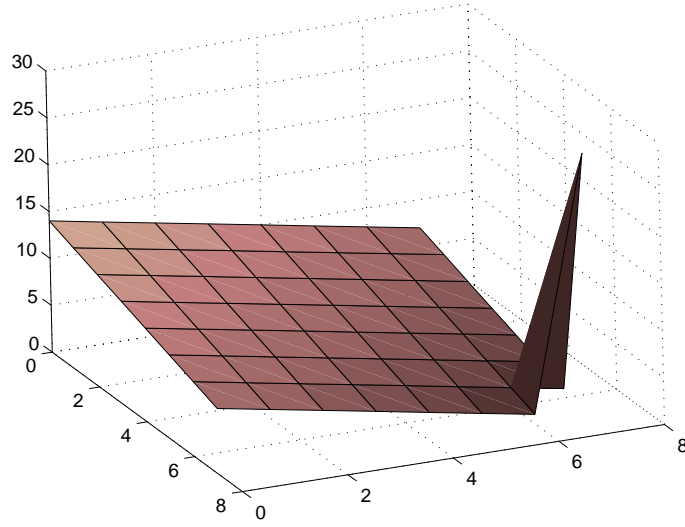


Figure 4.1: The suboptimum of $f_2(x_1x_2)$ is at $(0,0)$, but the optimum of $f_2(x_1x_2)$ is at $(7,7)$.

general GAs are not enough. Regarding related integers as a building block is a necessity to accomplish the task.

4.2 Experiments and Parameters

In our experiments, the genes of iECGA and simple GA are integers, and the genes of ECGA are binary strings. We use these algorithms to solve functions f_1 , f_2 , f_3 , and f_4 . Each experiment is conducted in 30 independent runs, and the average fitness is reported.

Because both ECGA and iECGA use tournament selection, we also use tournament selection in the simple GA. The tournament size is 32 in all experiments. Reported in many empirical studies, GA with uniform crossover has the best performance, so we use uniform crossover in the simple GA. Because of the memory limitation, the cardinality is 16 for f_1 and f_2 , 8 for f_3 , and 4 for f_4 .

The maximum detectable length of building blocks in ECGA is $\log_2 \mu$, where μ is the population size. The length in bit of building blocks in our test problems are 8 or 9. In all the experiments, the population size is 70,000, so the maximum detectable length is 16 bits. Three algorithms are executed up to 15 generations equivalent to 1,050,000 function evaluations.

4.3 Experimental Results

In all of the test functions, the second best solution has a fitness value half of that of the best solution. If the fitness value of the optimal solution is 20, the fitness value of the second best solution is 10. We present the results of experiments as the proportion to the maximum fitness, that is, we normalize the fitness to the range from 0 to 1. The scaled fitness value of the optimal solution is 1, and the scaled fitness value of the second best solution is 0.5.

The result of experiment one was shown in Figure 4.2. For function f_1 , three algorithms perform perfectly and all have fitness 1.0 for chromosomes of all lengths.

In Figure 4.3, we can see that ECGA has problems finding the optimal solution. For all chromosome lengths, ECGA can only find the second best solutions. The best and average fitness values are all 0.5. When the chromosome length is smaller than 60 integers (240 bits), iECGA and GA perform perfectly, but the performance of the simple GA decays quickly when the chromosome gets longer. When the chromosome length comes to 100 integers (400 bits), iECGA can find the optimal solution for most of the runs, but GA cannot.

The result of experiment four was shown in Figure 4.5. The result is almost the same as that of experiment two, except the performance of GA decays earlier. When the chromosome length is larger than 44 integers (88 bits), GA cannot find the optimal solution. The performance of ECGA is still much worse than that of the other two algorithms.

The result of experiment three was shown in Figure 4.4. The performance of iECGA is not as good as in the other three experiments but still outperform the other two algorithms.

The convergence analysis was shown in Figure 4.6 and Figure 4.7. The speed of convergence of iECGA is much faster than that of GA. In f_1 , iECGA converges after 700,000 function evaluations, but GA cannot converge even after 1,050,000 function evaluations.

function	d	BB	d^{BB}
f_2	16	2	256
f_3	8	3	512
f_4	4	4	256

Table 4.1: The cardinalities (d) and the length of building blocks (BB) of f_2 , f_3 , and f_4

4.4 Discussion

iECGA and GA are both operate in that integer domain, but why iECGA performs better than GA? The concepts of genetic linkage and building blocks are important components for GAs to solve problems. The main difference between iECGA and GA is the capability of detecting building blocks and genetic linkage. If the linkage configuration we find is correct, we may expect “good” building blocks will be preserved and “bad” building blocks will be weeded. Hence, iECGA performing better than GA is not unexpected.

ECGA is reliable and efficient in the binary domain, but why ECGA fails in the integer domain? If ECGA wants to find the linkage between integers, it has to consider several bits as one integer, and then consider several integers as one building block. That is, ECGA has to find building blocks of different hierarchies. It is the first difficulty.

The second difficulty is the selection of coding schemes. Most of GA users employ two’s complement to represent an integer, but there are many other kinds of representation, like the gray code. If the linkage between integers can be detected at the bit level, we call that the linkage “propagates” to the bit level. Different representations have different linkage propagations. The linkage between integers may or may not be detected at the bit level. Thus, how to choose an appropriate chromosome representation is an essential issue for GA to succeed.

Because of these difficulties, using ECGA to solve integer problems oftentimes cannot satisfy GA users. When we have to solve integer problems, we should use a specialized algorithm. Merely encoding the solutions as binary strings might not be a good choice.

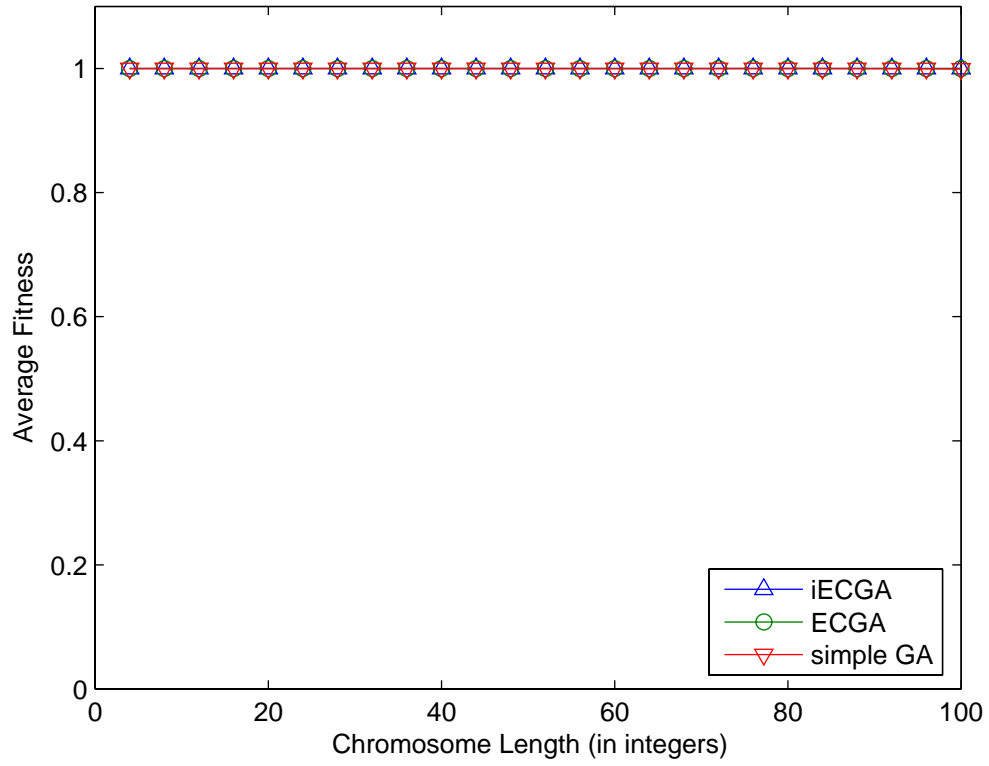
The convergence speed is an interesting property of ECGA and iECGA. Because they exchange building blocks but not genes, they avoid exchanging genes blindly. They converge more quickly than the simple GA does.

Another interesting observation is that ECGA and iECGA need sufficient individuals

to start the MPM step [14]. The population size has a direct ratio to d^{BB} , where d is the cardinality of an integer, and BB is the order of building blocks. In Table 4.1, we can see that the required number of individuals of f_3 is twice as large as that of f_2 and f_4 . Therefore, because the population size is not enough to start MPM, the performance of iECGA in f_3 suddenly goes down.



(a) The average fitness of iECGA, ECGA, and GA



(b) The best fitness of iECGA, ECGA, and GA

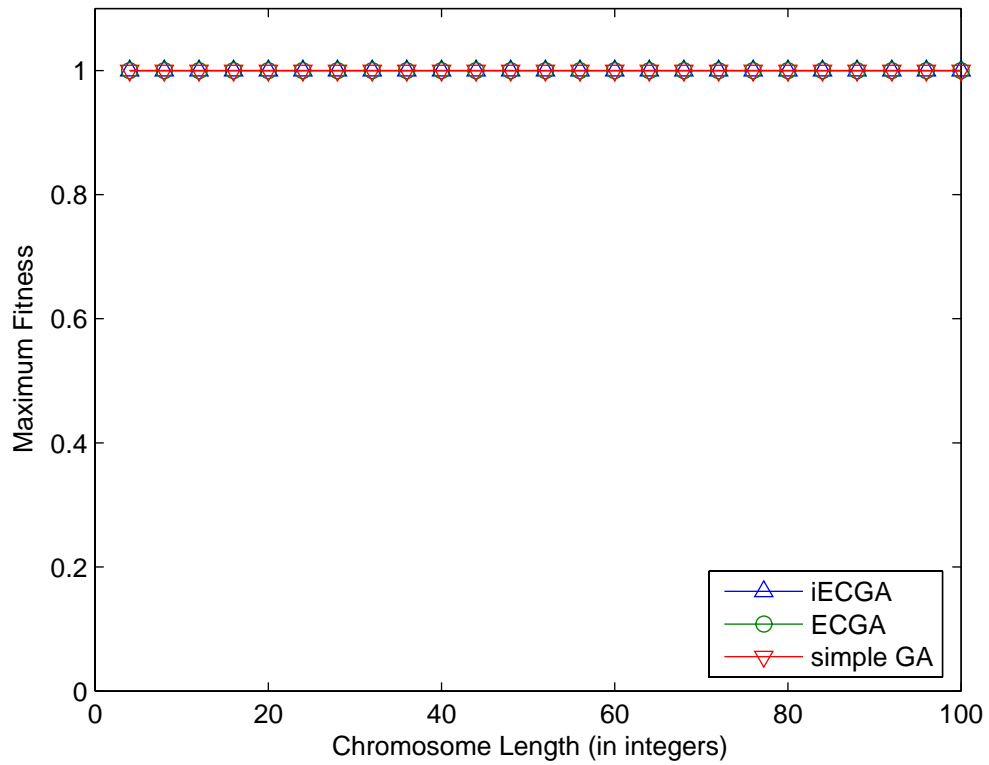
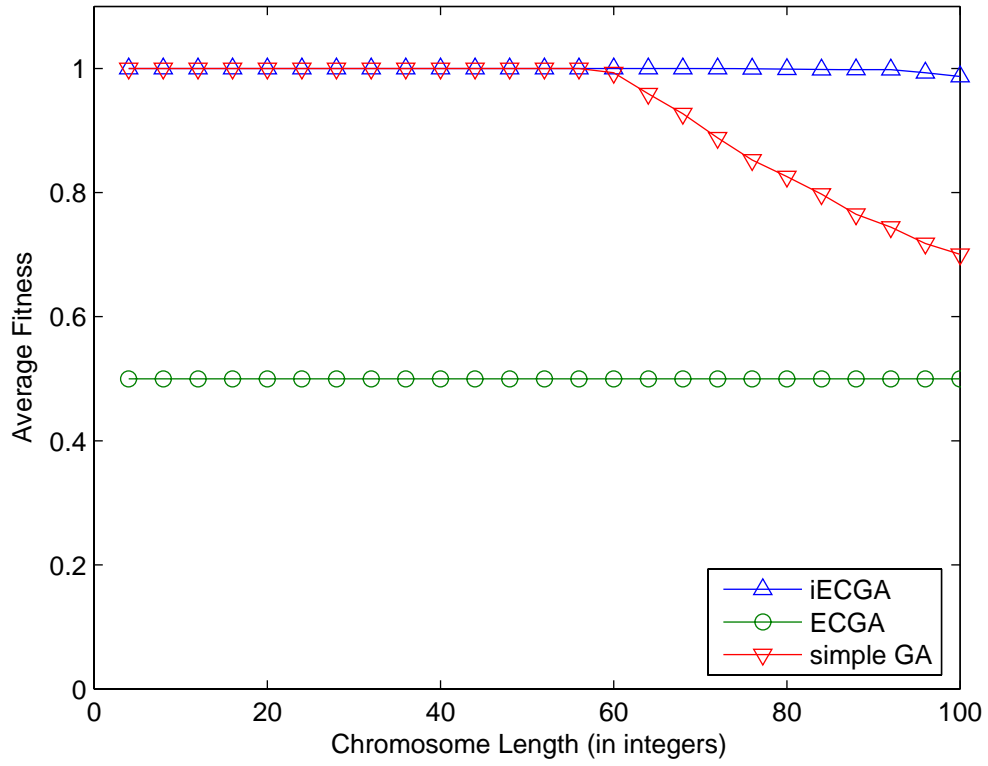


Figure 4.2: The average(a) and best(b) fitness of three algorithms in f_1 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

(a) The average fitness of iECGA, ECGA, and GA



(b) The best fitness of iECGA, ECGA, and GA

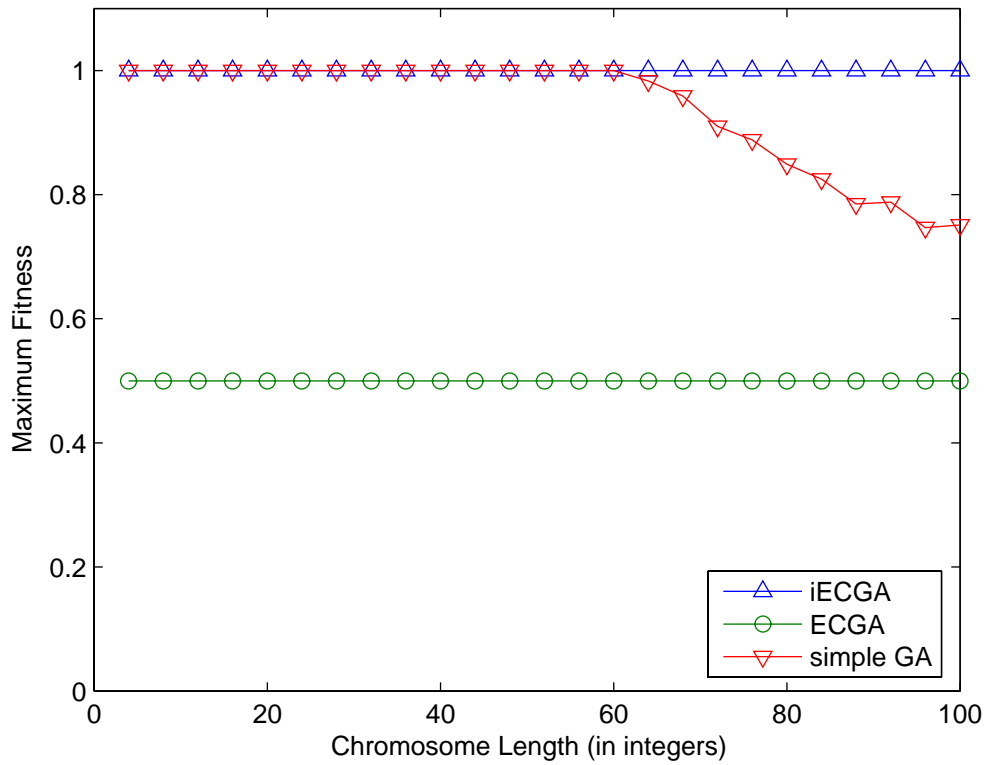
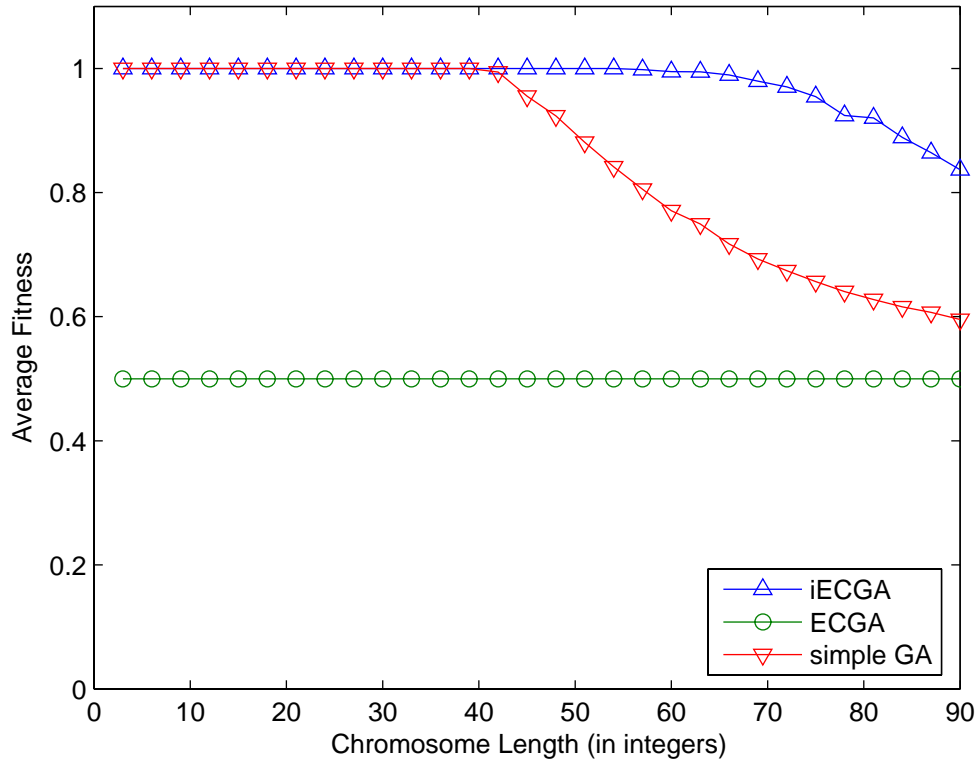


Figure 4.3: The average(a) and best(b) fitness of three algorithms in f_2 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

(a) The average fitness of iECGA, ECGA, and GA



(b) The best fitness of iECGA, ECGA, and GA

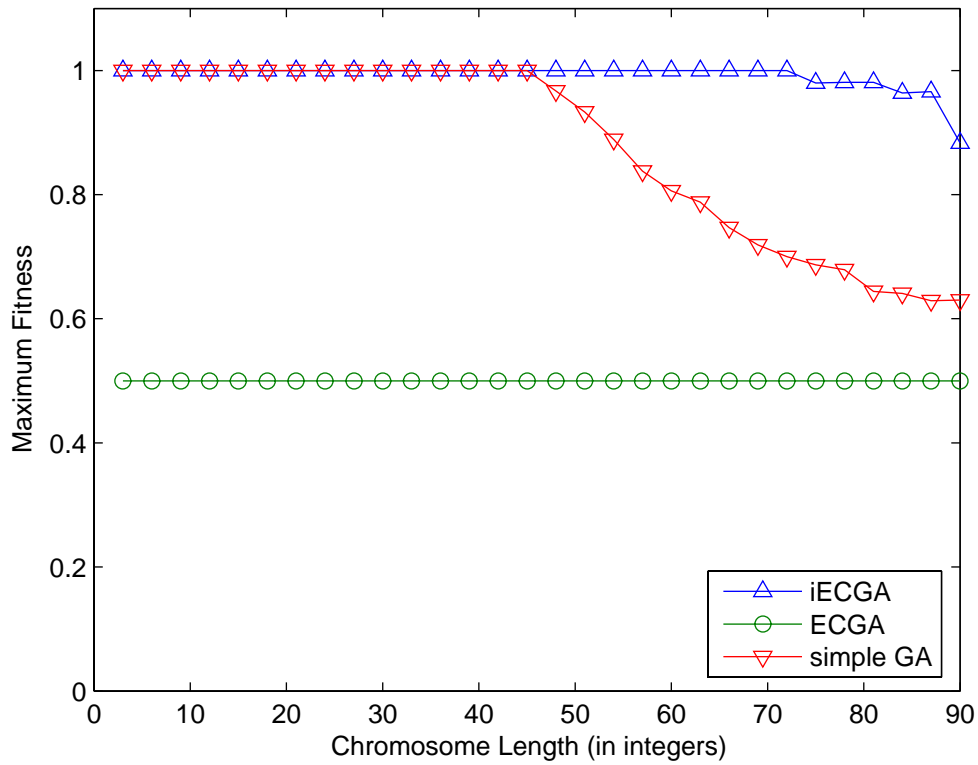
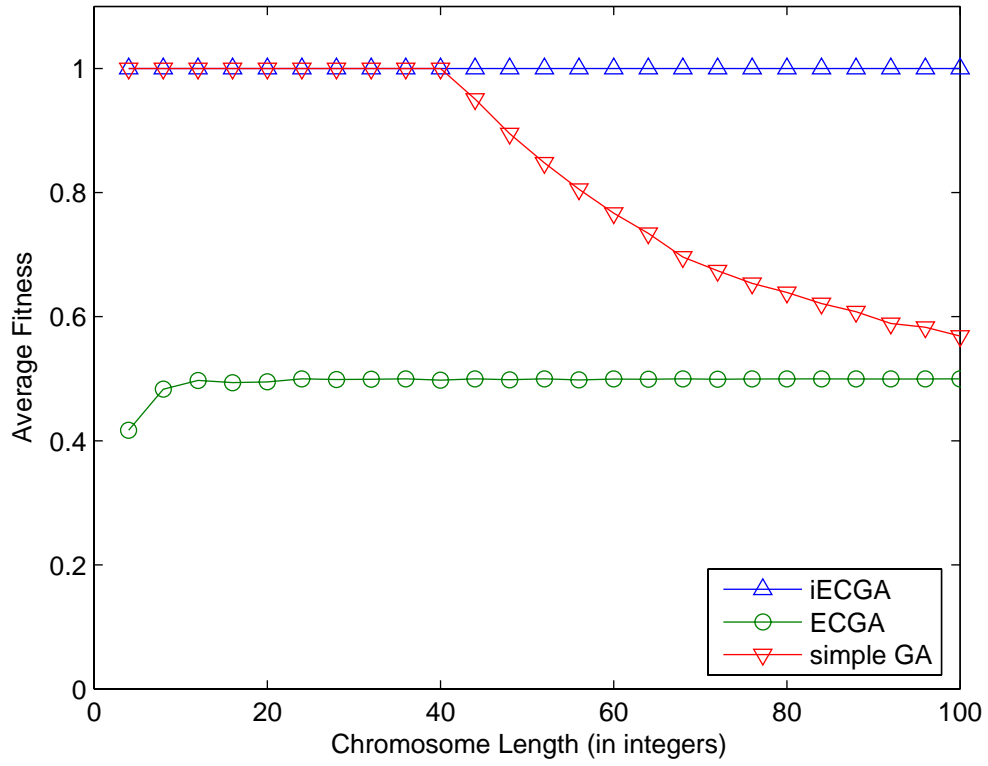


Figure 4.4: The average(a) and best(b) fitness of three algorithms in f_3 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

(a) The average fitness of iECGA, ECGA, and GA



(b) The best fitness of iECGA, ECGA, and GA

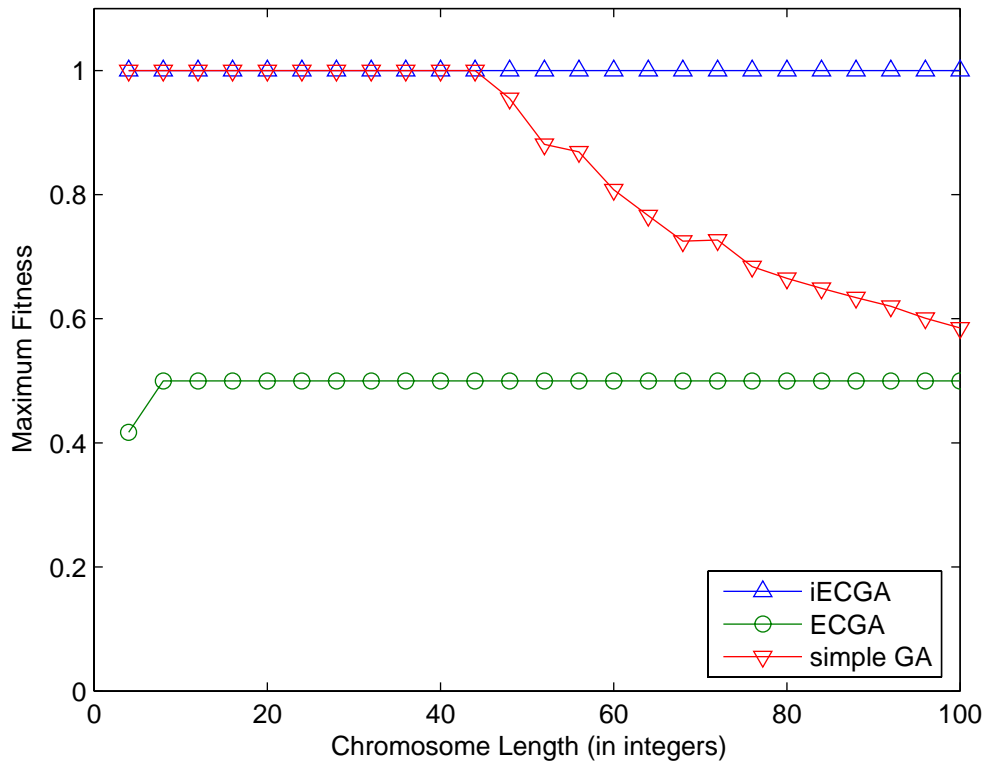
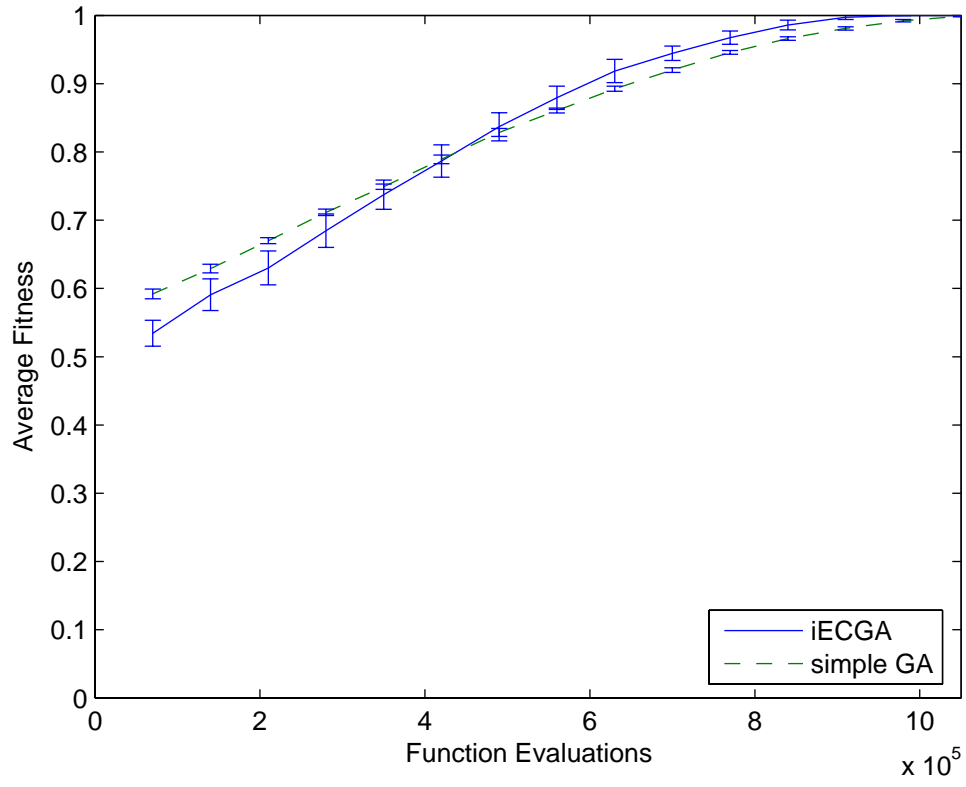


Figure 4.5: The average(a) and best(b) fitness of three algorithms in f_4 . X-axis is the length of a chromosome in the number of integers. Y-axis is the proportion to maximum fitness.

(a) Chromosome length 100 in f_1



(b) Chromosome length 100 in f_2

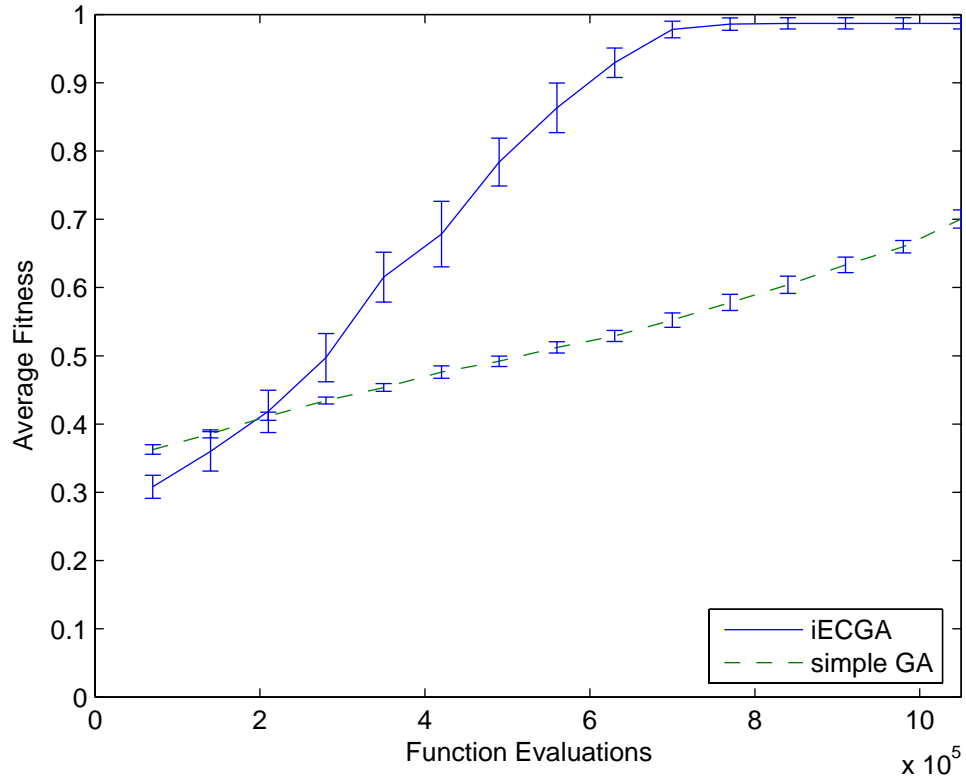
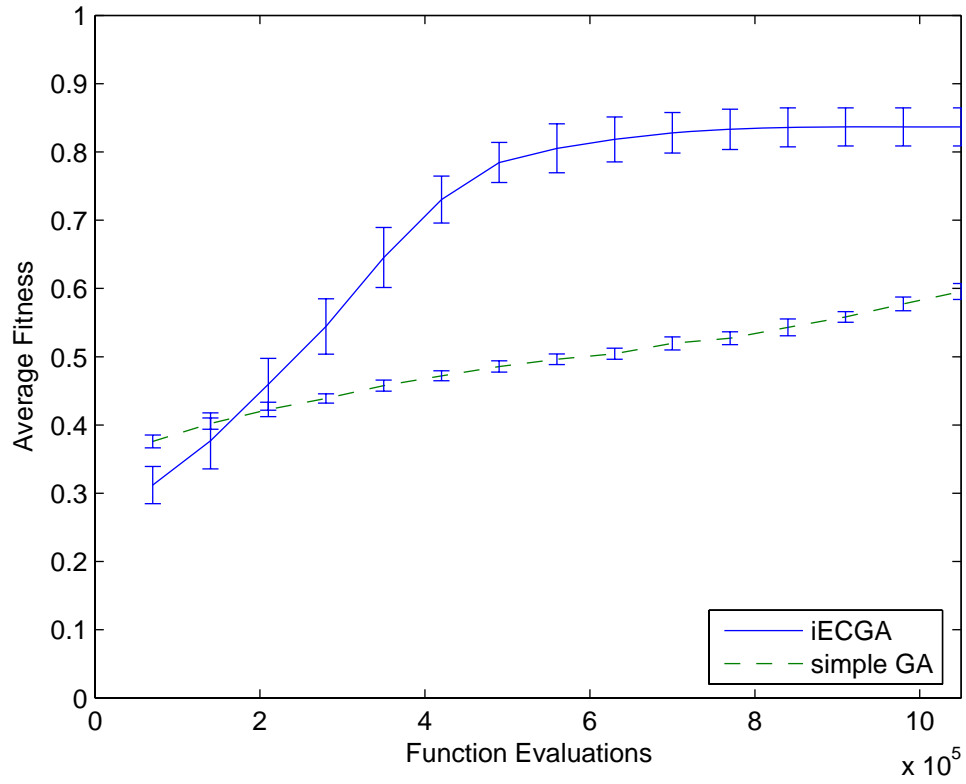


Figure 4.6: The convergence speed of f_1 and f_2

(a) Chromosome length 90 in f_3



(a) Chromosome length 100 in f_4

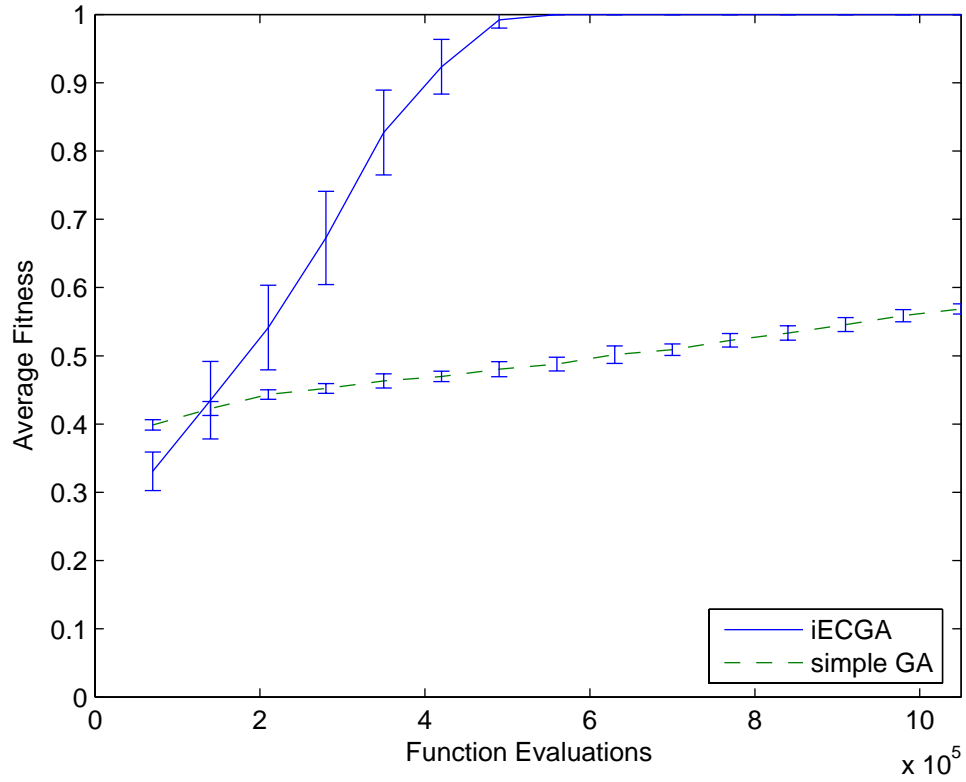


Figure 4.7: The convergence speed of f_3 and f_4

Chapter 5

Extend ECGA to Real-valued Domain

The real-coded ECGA is a new optimization framework, composed of the extended compact genetic algorithm [12] and split-on-demand (SoD) [13] method. In this section, we will show how SoD discretizes real numbers for ECGA and introduce the integration of ECGA and SoD.

5.1 Split on Demand for Discretization

Algorithm 2 Pseudo code for SoD.

```
procedure SPLIT-ON-DEMAND
    Split(lower_bound, upper_bound)
     $\gamma \leftarrow \gamma \times \epsilon$ 
end procedure

procedure SPLIT( $\ell$ ,  $u$ )
     $m \leftarrow \text{random}[\ell, u]$ 
     $N_\ell \leftarrow$  number of individuals in  $[\ell, m]$ 
     $N_u \leftarrow$  number of individuals in  $[m, u]$ 
    if  $N_\ell \geq N \times \gamma$  then
        Split( $\ell$ ,  $m$ )
    else
        Add a code for the range  $[\ell, m]$ 
    end if
    if  $N_u \geq N \times \gamma$  then
        Split( $m$ ,  $u$ )
    else
        Add a code for the range  $[m, u]$ 
    end if
end procedure
```

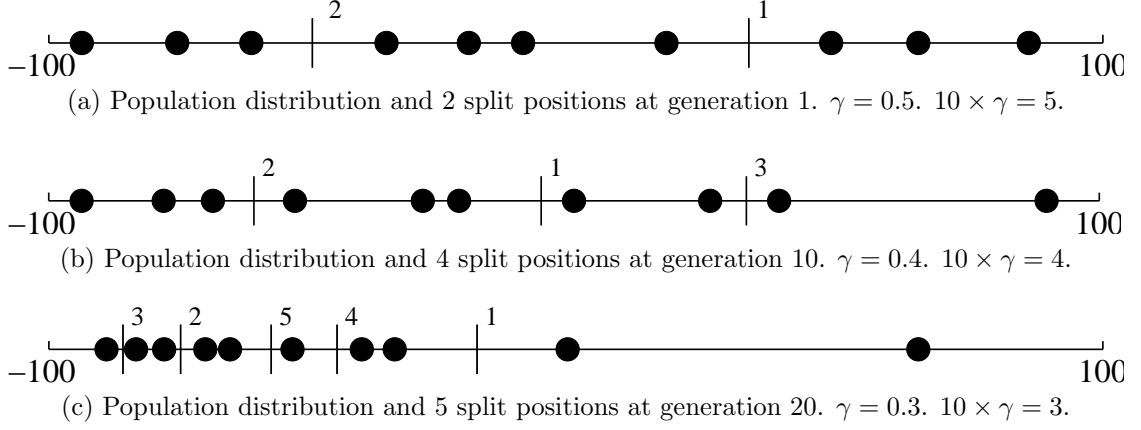


Figure 5.1: Populations and possible split positions (vertical lines). The numbers close to the positions are the order in which the positions are decided.

ECGA is designed for handling problems in the discrete domain. In order to employ ECGA to tackle problems in the continuous domain, certain mechanism is needed to transform the type of variables. In this work, we adopt an adaptive discretization technique, called split-on-demand (SoD) [13], to encode the individuals as real vectors into the ones as binary strings such that ECGA can accomplish the optimization task without significant modifications.

The main idea of SoD is to split the interval where we demand to obtain more information in order to build a more accurate probabilistic model for the region. There are two parameters for SoD: the *split rate*, γ , and the *split rate decay*, ϵ . γ is used to determine whether or not an interval should be split. Assuming that the population size is N , if an interval contains more than or equal to $N \times \gamma$ individuals, this interval should be split into two small intervals at a random position. By adjusting the split rate, we can control the accuracy of the probabilistic model and the size of code table. Figure 5.1 illustrates a splitting process under different γ . In Figure 5.1a, $\gamma = 0.5$ and the search space is split into three intervals. In Figure 5.1b, because γ gets smaller, the search space is split into more intervals.

Most good optimization algorithms consist of two elements: exploitation and exploration. In the proposed framework, we control the degree of exploitation vs. the degree of exploration by adjusting the split rate γ . We use a decreasing factor: ϵ , where $0 < \epsilon < 1$ to manipulate γ . At the early stage of search, we need more exploration than exploita-

tion. γ is set to 0.5, which means that one dimension of the search space will be split into only two or three intervals. As the search process goes, exploitation is more and more important. We multiply γ with ϵ at each generation to make it gradually smaller and smaller, and the MPM model is more and more accurate for the regions filled with individuals. Finally, Algorithm 2 shows the pseudo code for SoD.

5.2 ECGA with SoD

With the help of SoD, the real-coded ECGA (rECGA) can now handle problems in the continuous domain. The population in rECGA is represented in two forms: real vectors and binary strings. In the evaluation and selection phases, the population is in the form of real vectors. In the modeling and crossover phases, the population is in the binary-string form. SoD transforms real vectors into binary strings, and binary strings are converted back to real vectors by using random sampling. For example, if the code of an individual is 11 in binary, and the interval for the code 11 is $[-50, 0]$, the value is uniformly randomly sampled in the interval $[-50, 0]$. Finally, the integration of ECGA and SoD, which is the proposed framework in this paper, is shown in Algorithm 3.

Algorithm 3 Pseudo code for the real-coded ECGA.

procedure RECGA

$Gen \leftarrow 1$

 Initialize N individuals of real-numbers at random

while $Gen \leq Gen_{max}$ **do**

 Evaluate the population of size N

 Perform tournament selection of size S

 Use SoD to produce the code table

 Encode the population by using the code table

 Model the encoded population with MPM search

 Perform crossover with the given MPM model

 Generate the offspring with the code table

$Gen \leftarrow Gen + 1$

end while

end procedure

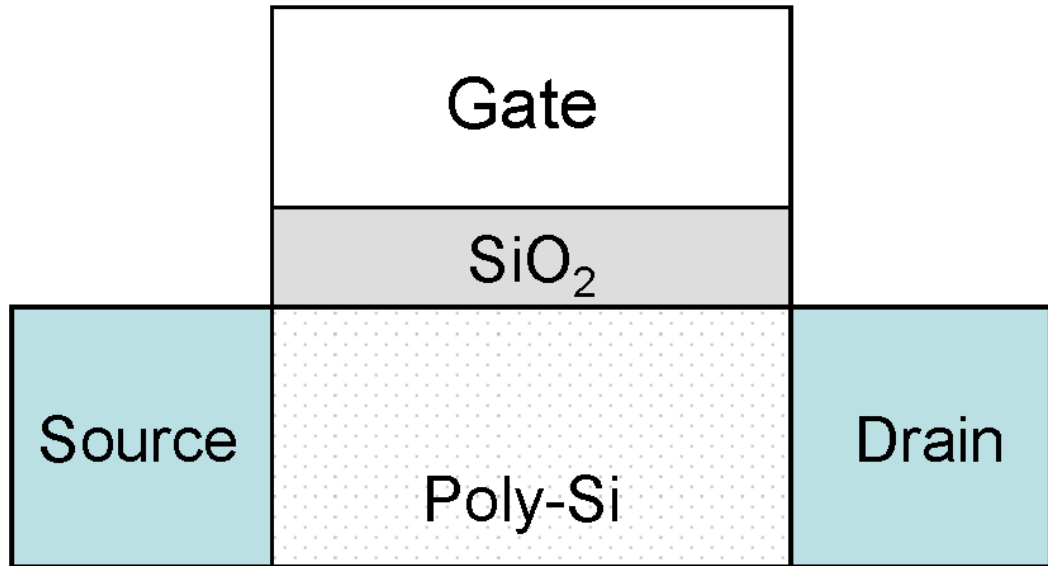
Chapter 6

Apply rECGA on Characteristic Determination Problem

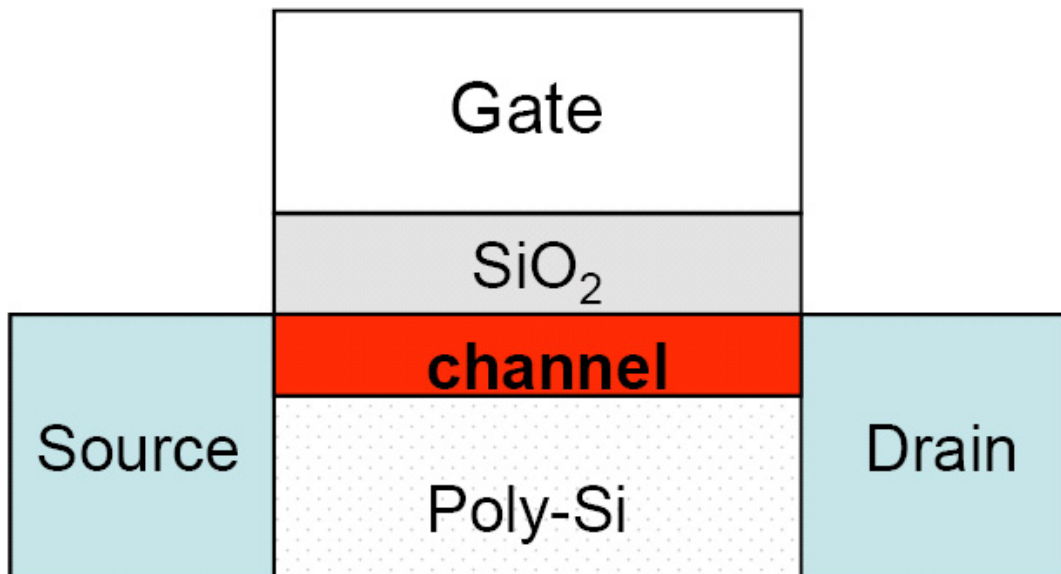
In the previous section, we proposed a new optimization framework in order to handle the characteristic determination problem for solid state devices. In this section, we apply the proposed framework to tackle three characteristic determination problems which we encountered while conducting research on developing thin-film transistors (TFT). The first one is to determine the quality parameters of the poly-Si thin-film under the normal condition, and the second one deals with different materials and fabrication processes under high gate bias. Finally, the third case is to determine the frequency response property of the solid state device.

6.1 Conventional TFT

A conventional poly-Si thin-film transistor, as shown in Figure 6.1a, is composed of three terminals: gate, source, and drain. When the transistor is turned on, electrons will transport from source to drain through the poly-Si area (the dotted area in the figure), and a high-conducting channel will be formed on the top of this poly-Si area, as shown in Figure 6.1b. The poly-Si area can exhibit a wide range of thin-film qualities. For a high quality poly-Si film, electrons can easily transport through it. As a result, the transistor can provide a large conduction current. For a low quality poly-Si thin-film, on the contrary, the electrical conductivity is low and the transistor output current is also reduced. Therefore, controlling the quality of the poly-Si film is essential to the creation and production of high-performance transistors. In addition to the output current, the



(a) Structure of poly-Si TFT.



(b) High-conducting channel for TFT.

Figure 6.1: The structure and the high-conducting channel formed for the conventional poly-Si TFT.

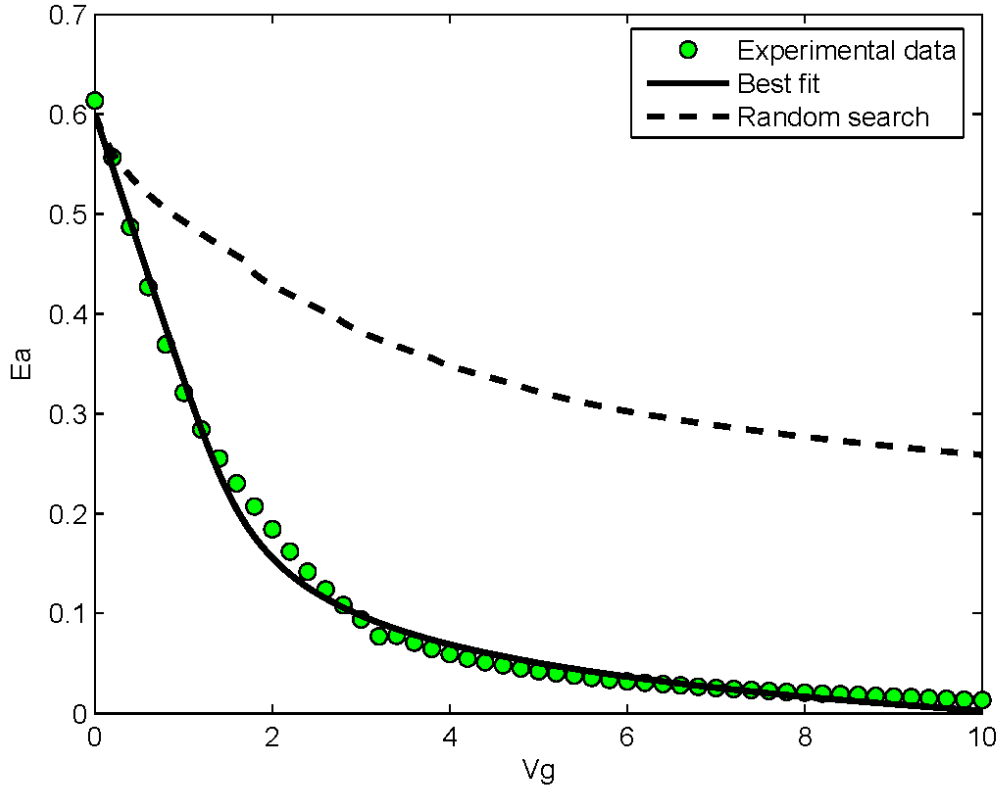


Figure 6.2: Experimental data and the match results for study case I.

Algorithm 4 Pseudo code for the fitness function in study case I

procedure F1

Input: $gene[0 \dots 4] = \{N_d, S_d, E_{td}, N_t, E_{tt}\}$

Input: experimental data $E_a[0 \dots 100]$, $V_G[0 \dots 100]$

Output: f - the fitness of $gene$

Constants: $q = 1.6 \times 10^{-19}$, $C_{ox} = 7 \times 10^{-8}$, $E_c = 1.2$

$i \leftarrow 0$, $f \leftarrow 0$

while $i < 101$ **do**

Fix $V_G = V_G[i]$, use binary approximation to obtain the value of E_a in Equation (6.3)

$f = f + |E_a - E_a[i]|$

$i = i + 1$

end while

return f

end procedure

quality of the poly-Si thin-film in the device is also a key issue to design the fabrication process and to develop the physical model as well as the SPICE model for poly-Si TFTs.

To characterize the poly-Si thin-film quality, the *defect state distribution*, $N(E)$, as

follows is usually utilized.

$$N(E) = \frac{N_d}{\sqrt{2\pi}S_d} \exp\left(-\frac{(E - E_{td})^2}{2S_d^2}\right) + N_t \exp\left(-\frac{E_c - E}{E_{tt}}\right), \quad (6.1)$$

where parameters N_d , S_d , E_{td} , N_t , and E_{tt} represent the properties of TFT. However, in practice, these parameters are not available and cannot be directly measured. Instead, these parameters have to be determined by measuring the observable experimental data and matching the equation

$$q \int_{E_c - E_{amax}}^{E_c - E_a} N(E) dE = C_{ox} (V_G - V_{fb} - \phi_s), \quad (6.2)$$

where $q = 1.6 \times 10^{-19}$, $C_{ox} = 7 \times 10^{-8}$, $E_c = 1.2$ are constants, and E_a , E_{amax} , V_G , V_{fb} , and ϕ_s are obtained from the experimental observation, to establish the relationship between the quality measurements $(N_d, S_d, E_{td}, N_t, E_{tt})$ and the observed outcomes $(E_a, E_{amax}, V_G, V_{fb}, \phi_s)$. After calculating the integral in Equation (6.2), we obtain

$$\frac{C_{ox} V_G}{q} = \left[\frac{-N_d S_d}{2} \text{Erf}\left(\frac{E_{td} - E}{\sqrt{2} S_d}\right) + N_t E_{tt} \exp\left(\frac{E - E_c}{E_{tt}}\right) \right] \Big|_{E_c - 0.6}^{E_c - E_a}, \quad (6.3)$$

where $\text{Erf}(\cdot)$ is the error function.

The characteristic determination problem in this case is to find the values of $N_d, S_d, E_{td}, N_t, E_{tt}$ according to the given set of measured values of E_a vs. V_G such that Equation (6.3) can be matched. We measured the value of E_a for $V_G = 0, \dots, 10.0$ for every 0.1, and obtained 101 pairs of (E_a, V_G) . The objective value for matching Equation (6.3) is defined as the

Parameter	Value
Population size (N)	250
Tournament size (S)	8
Number of generation (Gen_{max})	25
Crossover probability	0.975
Initial split rate (γ)	0.5
Split rate decay (ϵ)	0.995

Table 6.1: Parameters adopted in the real-coded ECGA for handling the three study cases.

sum of the absolute value of the differences between the calculated results and the 101 pairs of experimental data. A more clear procedure is shown in Algorithm.

Moreover, the ranges of the parameter can be decided according to physical laws. In this case, the ranges for the five parameters are

- N_d : 10^9 – 10^{15} ;
- S_d : 10^{-2} – 10^0 ;
- E_{td} : 0.5–0.7;
- N_t : 10^{11} – 10^{17} ;
- E_{tt} : 0.05–1.0.

We ran rECGA with 250 individuals for 25 generations. Detailed parameters of rECGA are shown in Table 6.1. In the 50 independent trials, the curve generated from the best solutions is shown as the solid line in Figure 6.2. To simply verify that the results we obtained are not merely “lucky shots”, we also conducted the pure random search for $250 \times 25 \times 50$ function evaluations. The result for the random search is shown as a dashed line in the figure. As we can see in the figure, the curve generated the pure random search goes very far from experimental data. As a side note, the curve of a similar matching quality can also be manually obtained for about three to five person-days, while the proposed framework takes only minutes to finish all the 50 trials.

6.2 TFT under High Gate Bias

When the transistors are operated under high gate bias, it is reported that the interface, as shown in Figure 6.3, between the poly-Si and the gate insulator also has great influence on the output current. As a consequence, to determine the property of TFTs under high gate bias, an interface-state distribution is inserted into $N(E)$ to appropriately model the

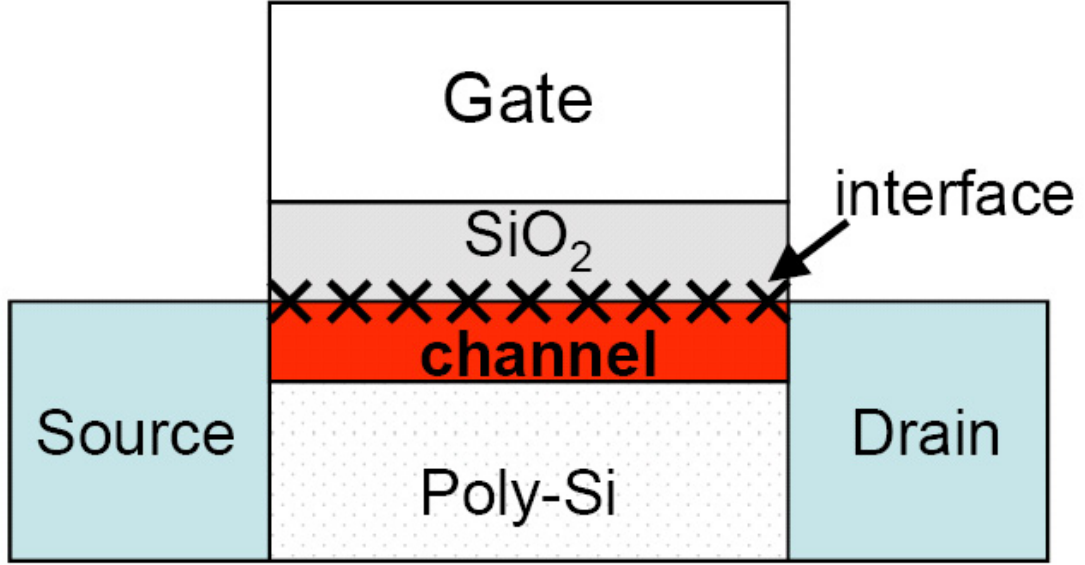


Figure 6.3: TFT under high gate bias.

overall defect quality:

$$N(E) = \frac{N_d}{\sqrt{2\pi}S_d} \exp\left(-\frac{(E - E_{td})^2}{2S_d^2}\right) + N_t \exp\left(-\frac{E_c - E}{E_{tt}}\right) + N_i \exp\left(-\frac{E_c - E}{E_{it}}\right), \quad (6.4)$$

where N_i and E_{it} are two more fitting parameters for the interface-state distribution. The ranges of N_i and E_{it} are

- N_i : 10^{11} – 10^{20} ;
- E_{it} : 0.05–1.0.

In this study case, we determine the quality parameters for four kinds of TFTs: ELA [22], FLA [23], SSL [24], and SPC [25]. There are several instances for each kind of TFT, and for simplicity in the present work, we choose only one or two instances to perform the computation. Similar to the previous study case, the values of the quality parameters can be obtained by fitting the experimental data to Equation (6.4). The fitness function F2 is similar as in study case 1. The pseudo code of F2 is shown in Algorithm.

The parameters for rECGA are identical to those used for case I, which is shown in Table 6.1. The curves generated by the best solutions in the 50 independent trials are

Algorithm 5 Pseudo code for the fitness function in study case II

procedure F2Input: $gene[0 \dots 6] = \{N_d, S_d, E_{td}, N_t, E_{tt}, N_i, E_{it}\}$ Input: experimental data $E_a[0 \dots K]$, $V_G[0 \dots K]$ Output: f - the fitness of $gene$ Constants: $q = 1.6 \times 10^{-19}$, $C_{ox} = 7 \times 10^{-8}$, $E_c = 1.2$ $i \leftarrow 0$, $f \leftarrow 0$ **while** $i < K + 1$ **do**(6.4) Fix $V_G = V_G[i]$, use binary approximation to obtain the value of E_a in Equation $f = f + |E_a - E_a[i]|$ $i = i + 1$ **end while**return f **end procedure**

shown as solid lines in Figure 6.5, and the best results obtained by the pure random search are shown as dashed lines. As we can observe in the figures, the pure random search can only match the first data point in all cases, while the proposed framework can provide high quality matching curves. Furthermore, these problem instances cannot be easily handled by human manipulation. We merely succeeded in manually matching a few problem instances for several person-weeks.

6.3 Frequency Response

The previous-addressed poly-Si thin-film quality and the interface quality also influence the frequency response of transistors. Since in circuitry, transistors may be operated under various frequencies, the frequency response is a very important property to determine the fabrication process, to determine the device model, and to determine the circuit design. In poly-Si TFTs, the frequency response is characterized through the capacitance measurement. As indicated in Figure 6.6a, the gate/SiO₂/poly-Si structure can be expressed by the equivalent circuit depicted in Figure 6.6b. That is, the total effective capacitance is the series of the oxide capacitance C_{ox} and the equivalent parasitic capacitance C_{eq} . The equivalent parasitic capacitance C_{eq} is the shunt of the bulk capacitance C_b and the interface capacitance C_{it} . Generally, C_{ox} is a constant, which is independent of gate bias or frequency, while C_b and C_{it} have a dependence on frequencies according to the following

equation

$$\begin{aligned} C_{eq} &= C_{it} + C_b \\ &= qD_{it} \frac{\tan^{-1}(\omega\tau_{it})}{\omega\tau_{it}} + qD_s \frac{\tan^{-1}(\omega\tau_s)}{\omega\tau_s}, \end{aligned} \quad (6.5)$$

where ω is $2\pi f$, and f is the frequency. D_{it} and τ_{it} are independent of frequencies, but depend on gate biases. D_s and τ_s are independent of both frequencies and gate biases, since the frequency and gate bias should not strongly influence the bulk properties.

As the previous two study cases, the frequency response parameters (D_{it} , τ_{it} , D_s , τ_s) cannot be directly measured, either. As a result, we measure C_{eq} under various gate biases and frequencies and determine the frequency response parameters according to the observed experimental data. The frequency response of C_{eq} under different gate biases are shown in Figure 6.7. The values of gate biases and frequencies we used to obtain the experimental data are

- Gate biases: $-1.3, -1.4, -1.5, -1.6, -1.7, -1.8, -1.9, -2.0, -2.1, -2.2$;
- Frequencies: $1 \times 10^4, 3 \times 10^4, 5 \times 10^4, 1 \times 10^5, 3 \times 10^5, 5 \times 10^5, 1 \times 10^6$.

There are totally 70 values for C_{eq} under the combinations of gate biases and frequencies measured. We used these experimental data to calculate C_{eq} according to Equation (6.5).

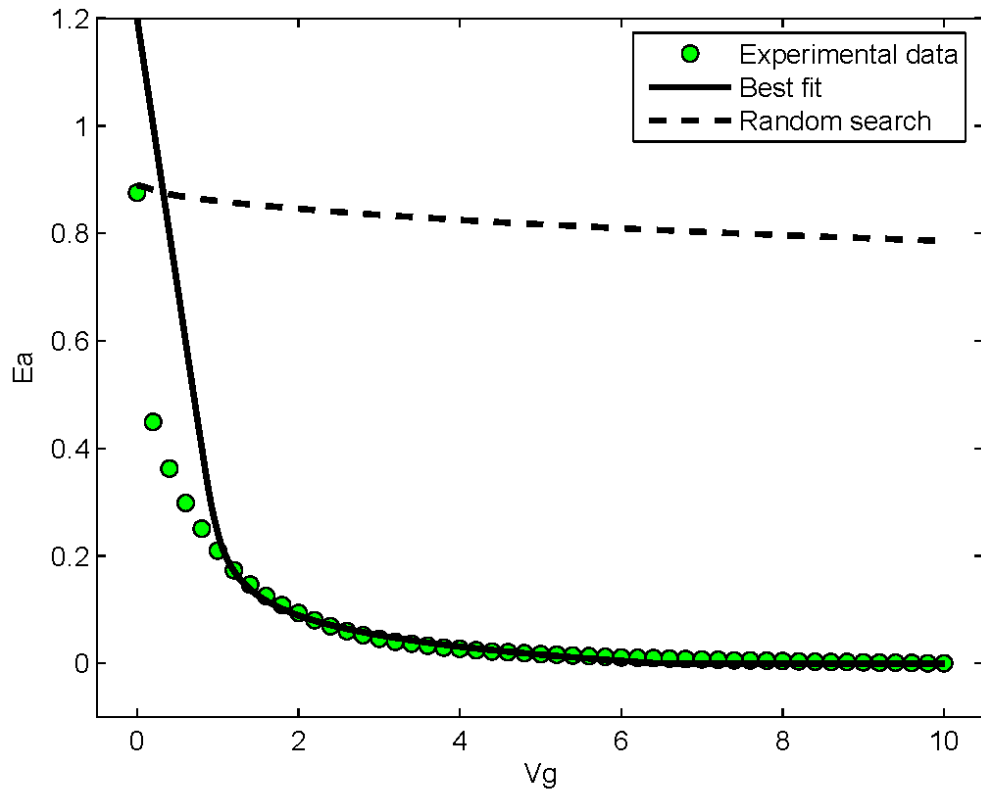
As shown in Figure 6.7, there are ten gate biases. Equation (6.5) indicates that there is a single pair of D_s and τ_s for all C_{eq} values, and for each set of C_{eq} values obtained under the same V_G , one pair of D_{it} and τ_{it} should be determined. Thus, there are 22 frequency response parameters. The objective value in this study case is also the sum of differences between the experimental data and the calculated results.

Without determining all the frequency response parameters simultaneously, we handle these parameters in separate groups. Because the values of C_{eq} are smallest when the gate bias is -1.3 or -1.4 , higher accuracy is needed to determine the parameters for the two sets of experimental data. Therefore, in the first group, we determine $D_s, \tau_s, D_{it}|_{V_G=-1.3}, \tau_{it}|_{V_G=-1.3}, D_{it}|_{V_G=-1.4}, \tau_{it}|_{V_G=-1.4}$. After obtaining D_s and τ_s , which are independent of V_G and f , we use the D_s and τ_s to determine D_{it} and τ_{it} for other gate biases. All the

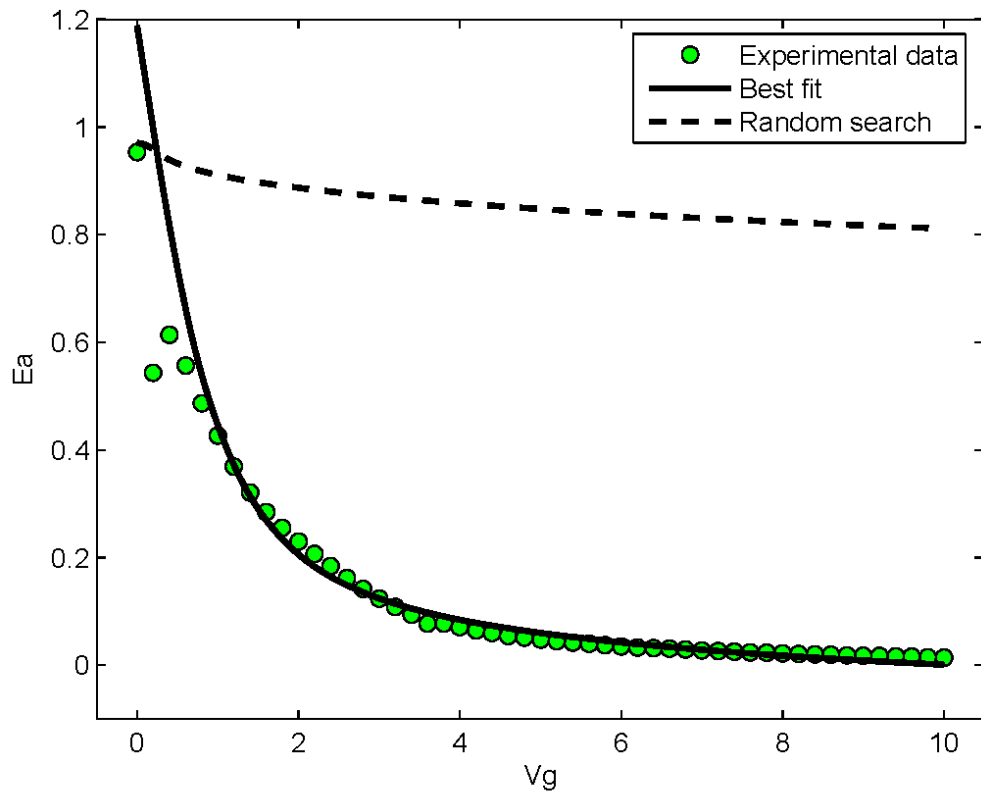
parameters of rECGA are identical to those in previous study cases, shown in Table 6.1, and the matching results are shown as the solid lines in Figure 6.8.

Figure 6.8 demonstrates that the matching results are remarkably satisfactory as the experimental data and the physical model pose a very difficult challenge for human to manually handle. Furthermore, based on the outcomes from the previous study cases, the pure random search has been decided inappropriate to handle the characteristic determination problem for solid state devices. As we can see in this work, the proposed framework of the real-coded ECGA, composed of ECGA and SoD, can be employed to tackle the characteristic determination problems of which the physical phenomena may be quite different.



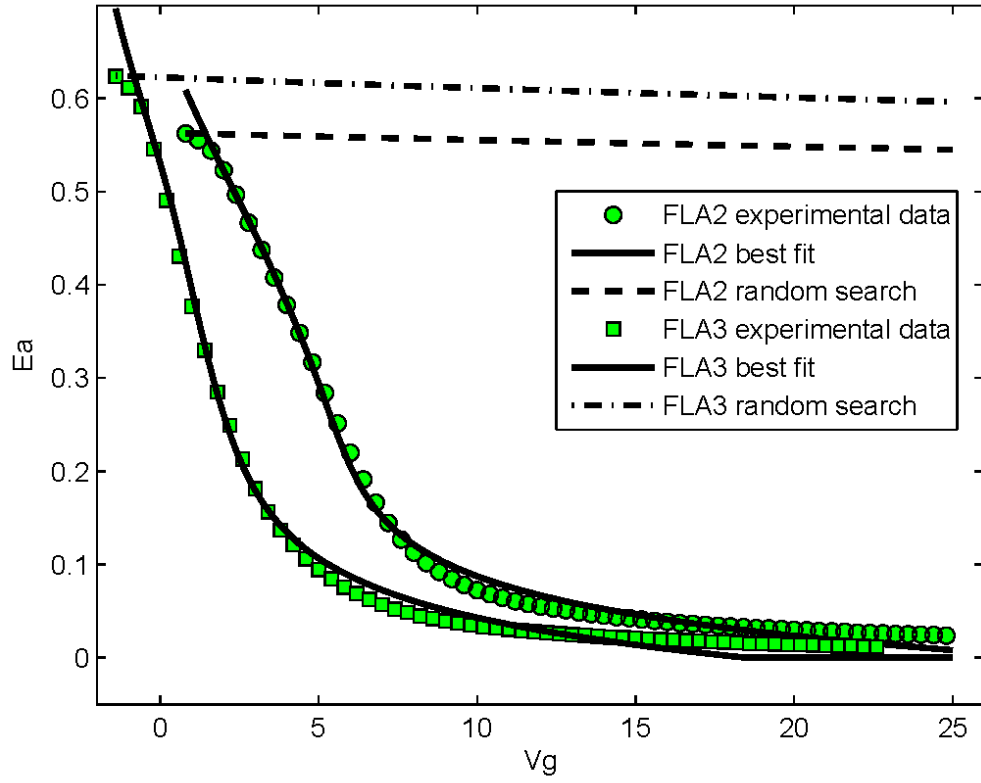


(a) ELA

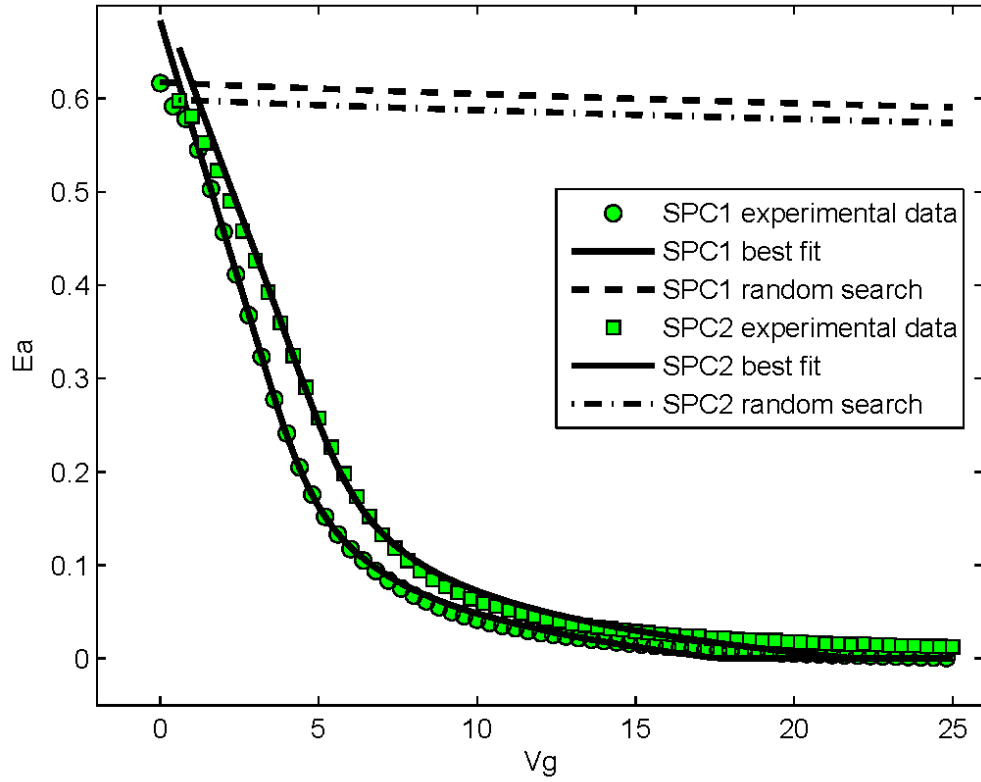


(b) SSL

Figure 6.4: Experimental data and the match results for study case II. ELA, SSL, FLA, and SPC are four different kinds of TFTs.

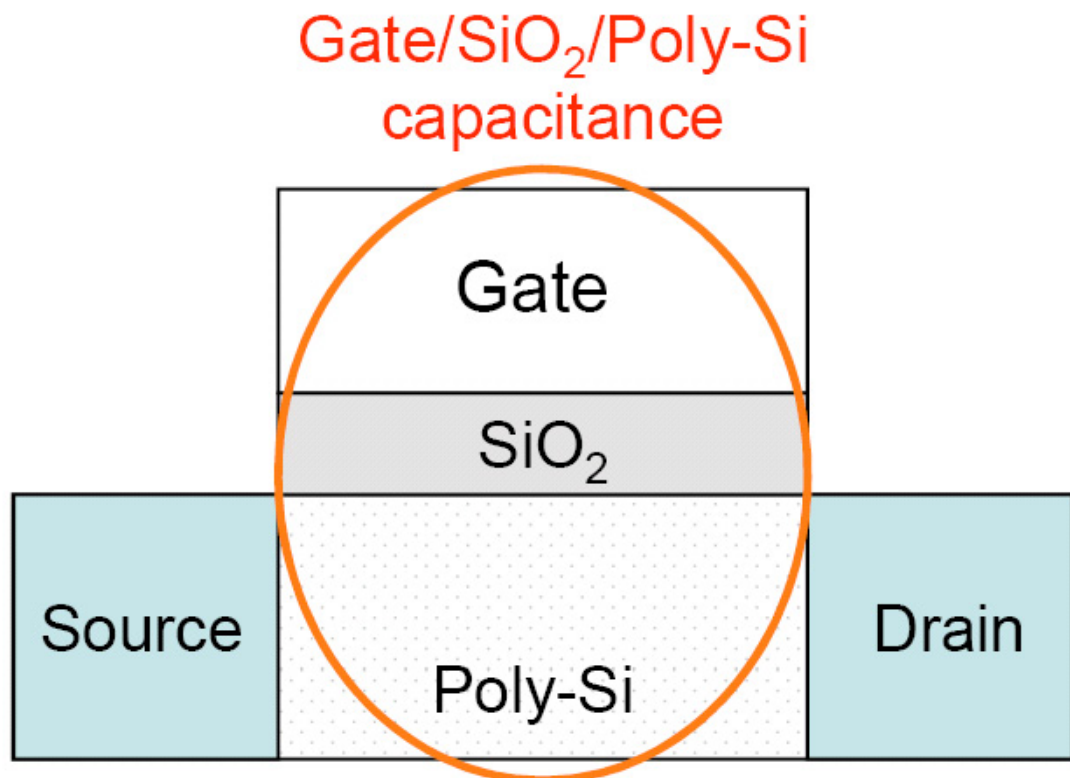


(a) FLA

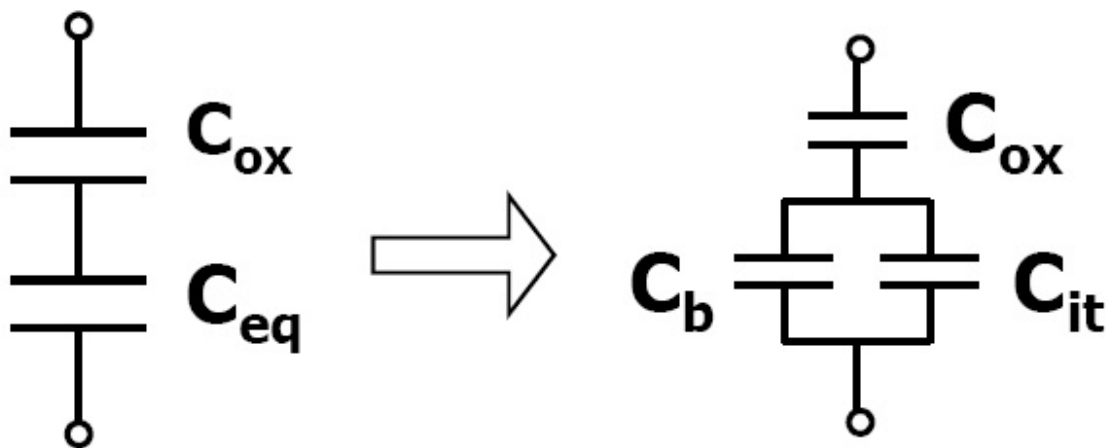


(b) SPC

Figure 6.5: Experimental data and the match results for study case II. ELA, SSL, FLA, and SPC are four different kinds of TFTs.

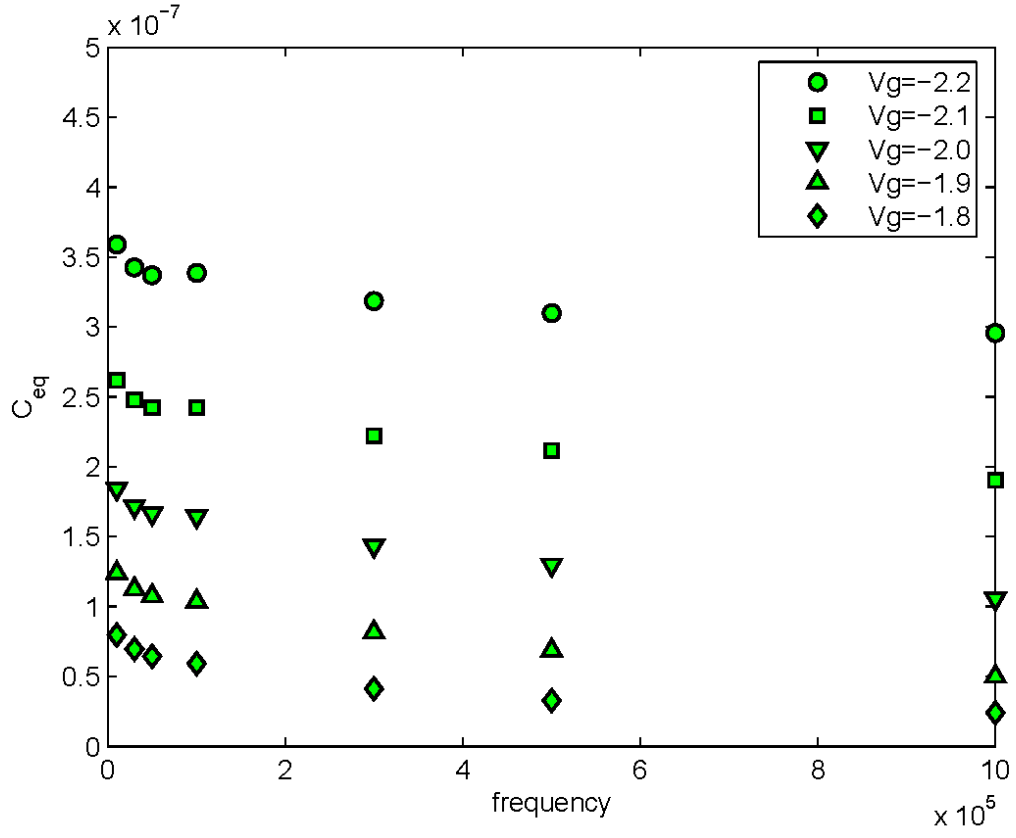


(a) Structure of gate/SiO₂/poly-Si.

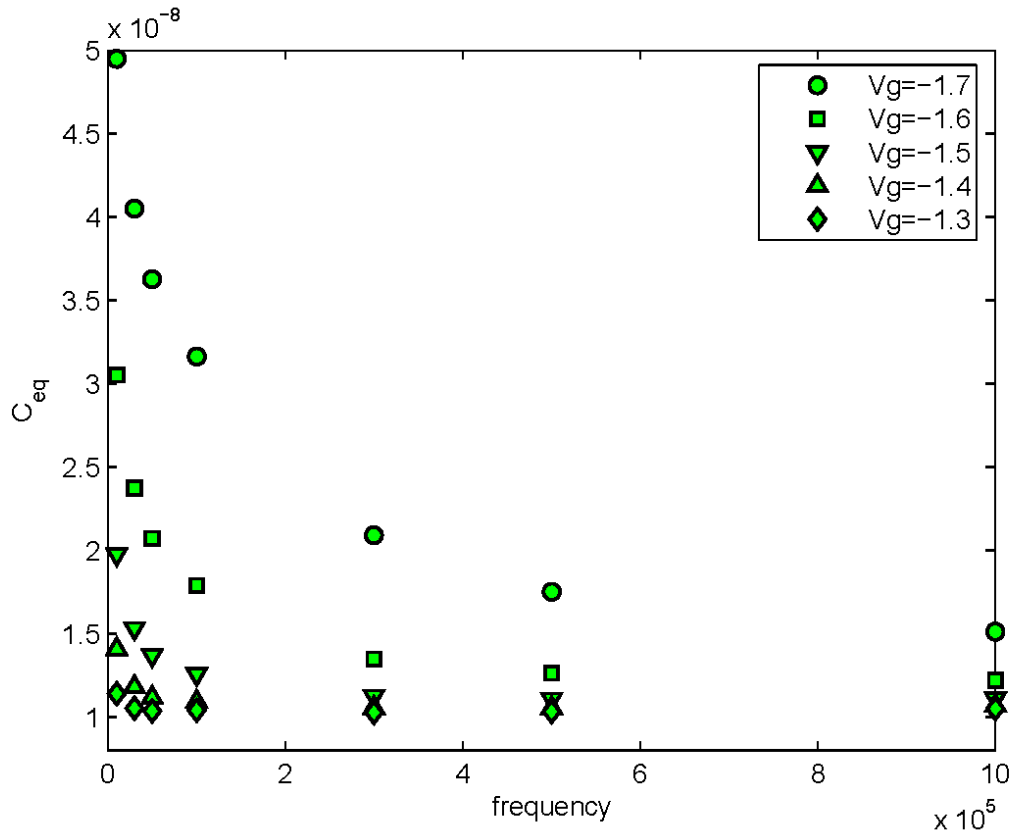


(b) Equivalent circuit for the structure.

Figure 6.6: Structure of gate/SiO₂/poly-Si and its equivalent circuit.

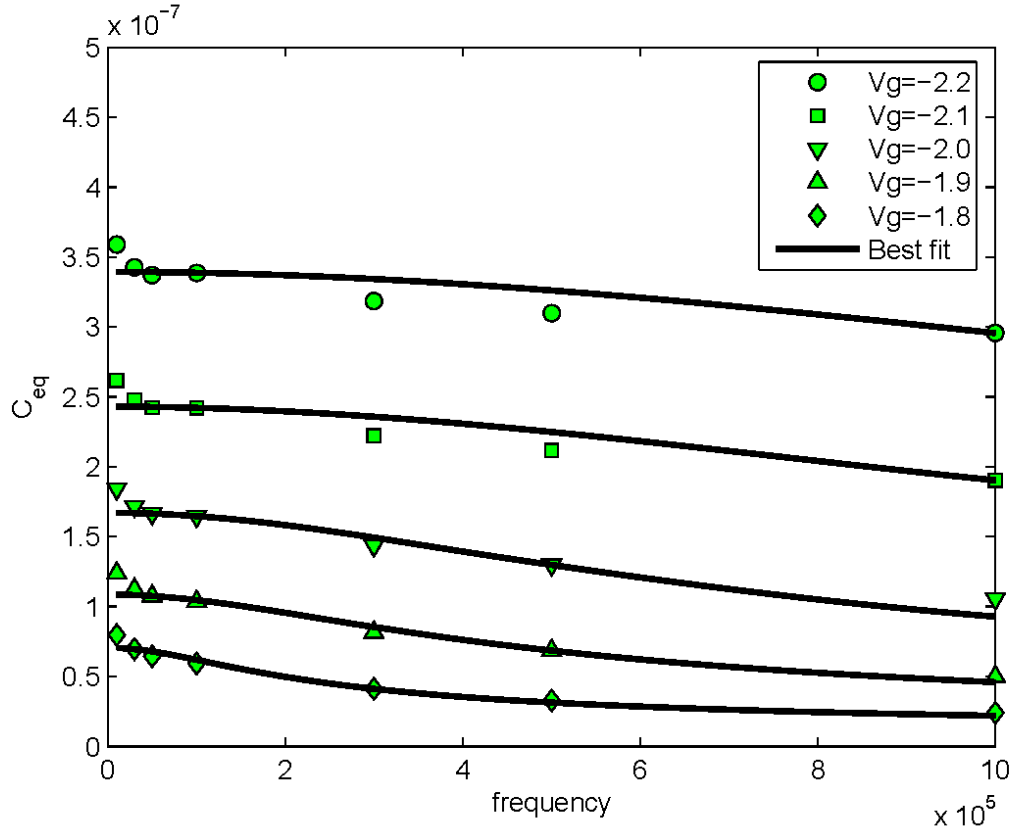


(a) Frequency response for $V_G = -2.2, \dots, -1.8$.

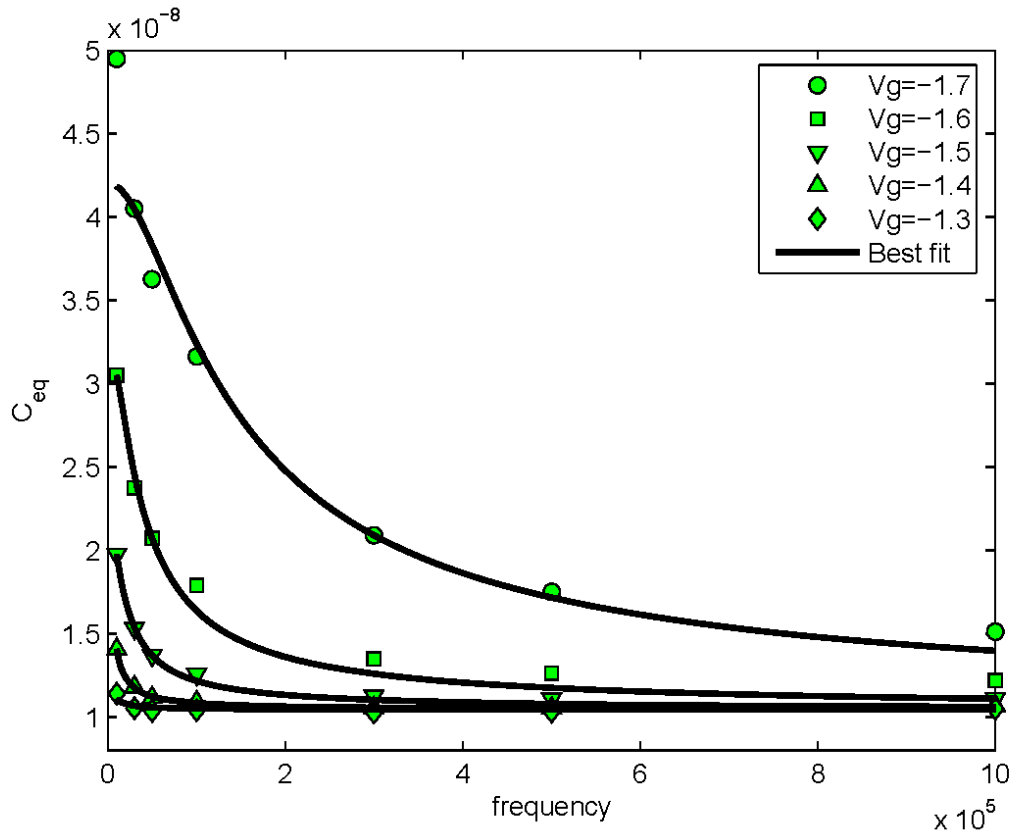


(b) Frequency response for $V_G = -1.7, \dots, -1.3$.

Figure 6.7: Frequency response for TFTs.



(a) Match results for $V_G = -2.2, \dots, -1.8$.



(b) Match results for $V_G = -1.7, \dots, -1.3$.

Figure 6.8: Match results for the frequency response.

Chapter 7

Conclusions

This chapter concludes this thesis. First we summarize our experiments, results, and conclusion. Then our long-term goals and objectives are proposed. Finally the main conclusions from this study are discussed.

7.1 Summary

In the paper, we study different kind of representations of ECGA. Our work has two parts. In the first part, we briefly reviewed the extended compact genetic algorithm (ECGA) and proposed iECGA, the integer extension of ECGA. The main difference between iECGA and ECGA is that they work in the different problem domains. iECGA can detect building blocks at the integer level but cannot find linkage at the bit level. In contrast, ECGA can successfully find linkage at the bit level, but fail to find hierarchical linkage in integer problems. For different types of problems, the appropriate algorithm should be selected to apply. That is, using right data type, GA and ECGA both work in the integer domain. According to the experimental results, iECGA outperforms GA when the problems have linkage of high order. But if the problem has no linkage between genes, GA and iECGA have the similar performance.

In the second part, we proposed a new optimization framework by integrating the extended compact genetic algorithm (ECGA) and split-on-demand (SoD), an adaptive discretization method, to tackle the characteristic determination problem for solid state devices. We employed rECGA to handle three characteristic determination problems of which the physical phenomena and the mathematical models were different. The nu-

merical results demonstrated that the proposed framework performed well on the study cases.

7.2 Future Work

Even the performance of iECGA achieves our expectation, there are several directions we can pursue in the future. iECGA avoid the problem of hierarchical linkage by encoding several bits as one integer but does not really solve the problem. How to find the hierarchical linkage is still a good question waiting to be answered.

Kumara and Goldberg have integrated ECGA with a mutation operator [26], iECGA can also be integrated with a similar mutation operator. Currently, iECGA can handle only the building blocks without overlap. The ability to handle overlapping building blocks can be developed in iECGA [27].

Representation is a fascinating topic in the field of evolutionary algorithms. The effect that represents an integer as a binary string is not clear yet. How the representation effects the linkage learning process is still a secret.

7.3 Main Conclusions

This study indicates the importance of using an appropriate data type to represent variables of different types, categories, or domains. Transferring or encoding the solutions may just introduce extra, unexpected difficulties to reduce the applicability and capability of existing good algorithms, instead of making the problem easier to solve. Therefore, we need to understand and investigate the algorithmic components much further in the future to design and develop better evolutionary algorithms.

When using ECGA to solve real-numbered problems, Split-on-Demand is a robust encoding method. The combined rECGA is a efficient and robust algorithm which we used to solve the characteristic determination problem. The characteristic determination problem is very important not only because the development of modern electronic computing equipment relies on solid state devices but also because more and more unknown physical phenomena are observed while the scale of the device gets smaller and smaller. In order

to gain understandings of all these unknown phenomena, getting access to the parameters that cannot be directly measured or observed is of great assistance. With the help of methodologies in evolutionary computation, this paper offers a good approach for researchers and developers to deal with encountered characteristic determination problems effectively and efficiently.

In the field of building-block research, more and more researchers pay attention on hierarchical BB problems. Several bits may compose of a small BB, but several small BBs may compose of a big BB. Traditional linkage learning schemes may fail when encountering hierarchical building-block problems. When the concept of linkage learning was proposed, representation is the focus of attention. When there is a bottleneck on study of representation, probability modeling becomes popular. This paper shows limitation of probability modeling and the possibility of right representation. Now it is time to concentrate our attention on representation. We wish our work can put some inspiration on the field of evolutionary computation.



Bibliography

- [1] J. H. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, 1975, ISBN: 0-262-58111-6.
- [2] G. R. Harik, “Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms,” Ph.D. dissertation, University of Michigan, 1997, also IlliGAL Report No. 97005.
- [3] Y. P. Chen, *Extending the Scalability of Linkage Learning Genetic Algorithm*. Springer, 2005.
- [4] D. E. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*, ser. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002, vol. 7, ISBN: 1-402-07098-5.
- [5] —, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Co., 1989.
- [6] P. Larranaga and J. A. Lozano, *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, ser. Genetic algorithms and evolutionary computation. Kluwer Academic Publishers, 2001, vol. 2.
- [7] M. Pelikan, D. E. Goldberg, and F. G. Lobo, “A survey of optimization by building and using probabilistic models,” *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.
- [8] S. ”Baluja, “”population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning,” , ”Pittsburgh,

- PA”, Tech. Rep. ”CMU-CS-94-163”, ”1994”. [Online]. Available: ”citeseer.ist.psu.edu/baluja94population.html”
- [9] M. Pelikan and H. Mühlenbein, ”Marginal distribution in evolutionary algorithms,” in *Proceedings of the International Conference on Genetic Algorithms Mendel ’98*. Czech Republic, 1998, pp. 90–95.
- [10] G. Harik, F. G. Lobo, and D. E. Goldberg, ”The compact genetic algorithm,” UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 97006, 1997.
- [11] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, ”BOA: The Bayesian optimization algorithm,” in *Proceedings of Genetic and Evolutionary Computation Conference 1999 (GECCO-99)*, 1999, pp. 525–532, also IlliGAL Report No. 99003.
- [12] G. Harik, ”Linkage learning via probabilistic modeling in the ECGA,” UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 99010, 1999.
- [13] C. H. Chen, W. N. Liu, and Y. P. Chen, ”Adaptive discretization for probabilistic model building genetic algorithms,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO)*, 2006, pp. 1103–1110.
- [14] K. Sastry and D. E. Goldberg, ”On extended compact genetic algorithm,” UIUC, Illinois Genetic Algorithms Laboratory, Urbana, IL 61801, USA, Tech. Rep. 2000026, 2000.
- [15] T. Mitchell, *Machine Learning*. McGraw Hill Text, 1997.
- [16] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. World Scientific, 1989.
- [17] D. H. Ackley, *A connectionist machine for genetic hillclimbing*, ser. Kluwer International Series In Engineering And Computer Science. Norwell, MA, USA: Kluwer Academic Publishers, 1987, vol. 28.
- [18] K. Deb and D. E. Goldberg, ”Analyzing deception in trap functions.” in *FOGA*, 1992, pp. 93–108.

- [19] D. E. Goldberg, B. Korb, and K. Deb, “Messy genetic algorithm: Motivation, analysis, and first results,” *Complex Systems*, vol. 3, pp. 493–530, 1989.
- [20] P. C. Hung and Y. P. Chen, “iECGA: integer extended compact genetic algorithm,” in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 1415–1416.
- [21] D. E. Goldberg, K. Deb, and B. Korb, “Messy genetic algorithm revisited: Studies in mixed size and scale,” *Complex Systems*, vol. 4, pp. 415–444, 1990.
- [22] J. S. Im and H. J. Kim, “On the super lateral growth phenomenon observed in excimer laser-induced crystallization of thin Si films,” *Applied Physics Letters*, vol. 64, no. 17, pp. 2303–2305, April 1994.
- [23] J.-M. Shieh, Z.-H. Chen, B.-T. Dai, Y.-C. Wang, A. Zaitsev, and C.-L. Pan, “Near-infrared femtosecond laser-induced crystallization of amorphous silicon,” *Applied Physics Letters*, vol. 85, no. 7, pp. 1232–1234, August 2004.
- [24] H. W. Zan, C. Y. Huang, K. Saito, K. Tamagawa, J. Chen, and T. J. Wu, “The crystallization mechanism of poly-Si thin film using high-power Nd: YAG laser with Gaussian beam profile,” in *Proceedings of MRS Symposium: Amorphous and Polycrystalline Thin Film Silicon Science and Technology—2006*, vol. 910, 2006, p. N/A, (In press).
- [25] A. T. Voutsas and M. K. Hatalis, “Structure of AS-deposited LPCVD silicon films at low deposition temperatures and pressures,” *Journal of the Electrochemical Society*, vol. 139, no. 9, pp. 2659–2665, September 1992.
- [26] K. Sastry and D. E. Goldberg, “Designing competent mutation operators via probabilistic model building of neighborhoods,” in *GECCO 2004*, June 2004, pp. 114–125.
- [27] T.-L. Yu, D. E. Goldberg, A. Yassine, and Y.-P. Chen, “Genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm,” in *GECCO 2003*, July 2003, pp. 1620–1621.