1-1-2007

Application of evolutionary computing in the design of high throughput digital filters.

Payman Samadi
University of Windsor

Application of Evolutionary Computing in the Design of High Throughput Digital Filters

by

Payman Samadi

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2007

Your file Votre référence
ISBN: 978-0-494-35014-0
Our file Notre référence
ISBN: 978-0-494-35014-0

Abstract

In this thesis, the application of Evolutionary Computing in the design of high throughput digital filters is studied. Evolutionary Computing are a group of problem solving methods which are based on biological evolution, such as natural selection and genetic inheritance. From these problem-solving techniques, Genetic Algorithm (GA) and Immune Programming (IP) are chosen for Digital Filter design application.

We start this research by developing iterative design methodologies for Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) 1-D filters and also FIR and IIR Quadrature Mirror Filter (QMF) banks. The proposed methodologies consider phase linearity as another constraint for the design formulation. Several studies on the performance analysis of Genetic Algorithm are performed. The dependence of Mean Square Error (MSE) on population size and number of generations were investigated. The performance of GA over different cross-over techniques and different values for probability of cloning (P_c) and probability of mutation (P_m) is analyzed too.

Furthermore to obtain high throughput rate digital filters, Common Subexpression Elimination (CSE) is applied on the coefficients. In this thesis, the existing algorithm for 2 non-zero digits CSE in both vertical and horizontal position is extended to 3

non-zero digits for vertical elimination. At the end, a new algorithm for the design of high throughput digital filters with CSE constraint, linear phase characteristics and Canonical Signed Digit (CSD) coefficients is proposed which leads to more efficient digital filters.

To My Family.

Acknowledgments

I would like to express my sincere gratitude and appreciation to Dr. Majid Ahmadi, my supervisor, for his invaluable guidance throughout the course of this thesis work. Special thanks to Dr. Kemal Tepe, Dr. Huapeng Wu, Dr. Shervin Erfani and Dr. Esam Abdel-Raheem for their expert guidance and constant support throughout my study. I would also like to thank Dr. W. Abdul-Kader for reviewing this work.

I also sincerely appreciate my family for their endless support and my friends for their help and friendship.

Contents

List of Figures

List of Tables

List of Abbreviations

1-D	One-dimension
2-D	Two-dimension
AIS	Artificial Immune Systems
ASIC	Application Specific Integrated Circuits
CSD	Canonical Signed Digit
CSE	Common Subexpression Elimination
CI	Computational Intelligence
DSP	Digital Signal Processing
E	Set of Graph Edges
FDM	Frequency Division Multiplexing
FIR	Finite Impulse Response
FPS	fitness proportionate selection
G	Graph
GA	Genetic Algorithm
ID	Identification
IIR	Infinite Impulse Response
IP	Immune Programming
LP	Linear Phase
Pc	Probability of Cross-over
Pm	Probability of Mutation
Pi	Probability of Inversion
QMF	Quadrature Mirror Filter
TDM	Time Division Multiplexing
TSP	Traveling Salesman Problem

Chapter 1

Introduction

Interests on discrete time signals have been enormously increased during past three decades. In discrete time signals, error correction is much easier and higher transmission rates can be achieved. Digital Signal Processing (DSP) is a field of engineering that works on the discrete time signals. DSP has a wide range of applications, from home devices such as DVD players and TV tuners to precise biomedical devices (Magnetic Resonance Imaging (MRI)) and in security applications such as face and iris recognition.

One important branch of DSP is Digital Filters [36]. Digital Filters work on discrete time data and basically perform digital mathematical operations on them. These data can be one-dimension (1-D), two-dimensions (2-D) [47] or in general N-dimensions (N-D). As an example, 1-D signal can be audio signal, 2-D signal can be image of a scene and 3-D signal can be a video.

1.1 Digital Filters

Digital Filters are an important part of Digital Signal Processing. They can be used for different applications but in general they are utilized for separating the combined signals and for restoring the distorted signals. As an example when we have an audio signal which is corrupted with the high frequency noise, a low-pass filter with the cut-off frequency of $20KHz$ can cut the unwanted noise.

Digital Filters may be implemented in software or hardware. Software implementation can be a software on a general-purpose computer and hardware implementation can be a DSP chip or an ASIC processor.

Similar to discrete time data, Digital Filters can be in one-dimension (1-D), two-dimensions (2-D) and in general in N-dimensions (N-D) [40]. In this section different kinds of (1-D) digital filters are reviewed.

1.1.1 One-Dimensional Filters

1-D digital filter can be in the form of FIR (non-recursive) or IIR (recursive) and can be characterized by their difference equation. A linear, time-invariant FIR filter can be shown as:

$$y(n) = \sum_{i=0}^{N} a(i)x(n-i) \tag{1.1}$$

By using the z-transform, a digital filter can be described in z-domain which is helpful in frequency domain analysis. In theory, the transfer function of a digital filter is the z-transform of its impulse response. The transfer function of a linear, time-invariant FIR filter is defined as:

$$H(z) = \sum_{n=0}^{N} h(n)z^{-n} \tag{1.2}$$

The same formulation is also available for IIR filters. A linear, time-invariant IIR filter can be shown as:

$$y(n) = \sum_{i=0}^{N} a(i)x(n-i) - \sum_{i=1}^{M} b(i)y(n-i) \tag{1.3}$$

And its transfer function can be shown as:

$$H(z) = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} \tag{1.4}$$

In the digital filter design we are always interested to have stable filters. FIR filters are always stable therefore there is no need for stability check, but for IIR filters stability check should be performed. The poles of the transfer function determine whether the filter is stable or not. In Equation 1.4, the denominator polynomial $D(z)$ must satisfy the following constraint:

$$D(z) \neq 0 \;\; \forall |z| \geq 1 \tag{1.5}$$

By this constraint all poles of the $D(z)$ are located inside the unit circle of z plane, therefore the filter is stable. For stability check we have used the Jury-Marsden [40] method which does not require finding any polynomial roots and only requires the calculation of the determinant of a series of 2 by 2 matrices.

1.1.2 Quadrature Mirror Filter Banks

Quadrature Mirror Filter (QMF) banks have been subject of research for many years [48]. They are applied for the situation where a discrete-time signal $x[n]$ is needed to be split into a number of sub-band signals $x_i[n]$ so that each can be processed separately. Typical processing comprises down-sampling, coding for transmission and storage. Subsequently at some point, these signals are needed to be recombined together to have the original signal reconstructed. The most common

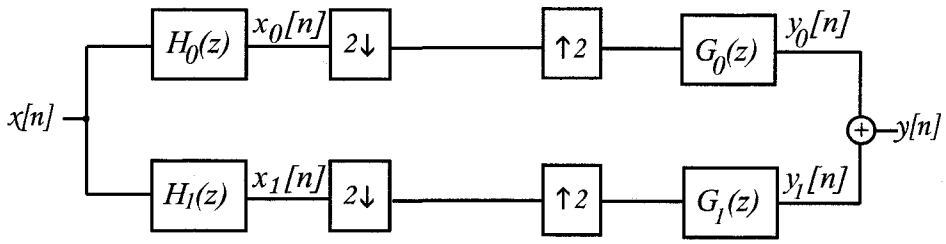


Figure 1.1: The two-channel QMF Bank.

applications are frequency domain speech samplers, sub-band coders for speech signals and digital trans-multiplexers used in Frequency Division Multiplexing (FDM) / Time Division Multiplexing (TDM) conversion.

Based on the type of synthesis and analysis filter (FIR or IIR) and number of channels, there are different kinds of filter banks. In this research we have focused on 2-channel QMF bank for both FIR and IIR. Fig. 1.1 shows the basic two-channel QMF bank. The analysis filter bank is composed of a low-pass filter $H_0(z)$ and a high pass filter $H_1(z)$. These two filters split the incoming signal to two frequency bands. The sub-band signals are then down-sampled by a factor of two. Each down-sampled sub-band signal is encoded by exploiting the special spectral properties of the signal, such as energy level and perceptual importance. At the receiving end these signals are up-sampled by a factor of two and fed into the two-band synthesis filter bank $G_0(z)$ and $G_1(z)$, and at the end the output is created by adding these two signals.

Up-sampling of $x_0[n]$ and $x_1[n]$ result in images, which have to be eliminated by $G_0(z)$ and $G_1(z)$. The output of $G_0(z)$ and $G_1(z)$ are a good approximation of $x_0[n]$ and $x_1[n]$, and the reconstructed signal $y[n]$ closely resembles $x[n]$. In Quadrature Mirror Filter banks the response of $H_0(z)$ is the mirror image of the response of $H_1(z)$.

In Fig. 1.1 the relation between the input and output signal is [40]:

$$Y(z) = \frac{1}{2}G_0(z)H_0(z) + G_1(z)H_1(z) + \frac{1}{2}G_0(z)H_0(-z) + G_1(z)H_1(-z)x(-z) \quad (1.6)$$

In order to have an alias-free QMF bank , we should set these two filters in a such way that aliasing effect be canceled; this will lead to Linear Time Invariant (LTI) system. Following is the condition of alias-free QMF bank [40]:

$$G_0(z) = H_1(-z) \quad (1.7)$$

$$-G_0(-z) = G_1(z) \quad (1.8)$$

$$H_1(z) = H_0(-z) \quad (1.9)$$

According to the above equation, by designing $H_0(z)$ only, we will have the whole system designed. For FIR QMF bank there is no need for stability check but for IIR filters, stability check must be carried out for the designed filters. We have used the Jury-Marsden [40] method, which determines the stability by the calculation of a series of 2 by 2 determinants and does not require finding any polynomial roots. The N^{th} order real coefficient IIR $H_0(z)$ is defined as:

$$H(z) = \frac{\sum_{i=0}^{N} a(i)z^{-i}}{\sum_{i=0}^{M} b(i)z^{-i}} = \frac{N(z)}{D(z)} \quad (1.10)$$

The group delay of a filter is a measure of the average delay of the filter as a function of frequency and is defined as:

$$\tau(w) = -\mathrm{Re}[z\frac{dH(z)/dz}{H(z)}]_{z=e^{jwT}} = \mathrm{Re}[z\frac{D'(z)}{D(z)} - z\frac{N'(z)}{N(z)}]_{z=e^{jwT}} \quad (1.11)$$

Where $D'(z) = \frac{dD(z)}{dz}$ and $N'(z) = \frac{dN(z)}{dz}$.

For the N^{th} order real coefficient FIR filter, $H_0(z)$ is

$$H(z) = \sum_{i=0}^{N} a(i)z^{-i} \quad (1.12)$$

1.2 CSD Coding

CSD number system is a representation of a number as a sum and difference of power of two.

$$x = \sum_{k=1}^{M} s_k 2^{-p_k} \tag{1.13}$$

Where s_k are ternary digits, $s_k \in \bar{1}, 0, 1$, and $\bar{1}$ is defined as -1.

M is a pre-specified word length, and $p_k \in 0, 1, ..., M$.

and with the constraint:

$$s_k \times s_{k+1} = 0, \text{ for all the } k \in 0, 1, ..., M. \tag{1.14}$$

As an example, we want to represent 0.8743 in CSD number system. There is limit of 4 non-zero digits and the word length is 8. The CSD representation of 0.8743 is:

$0.8743 \simeq 2^0 - 2^{-3} - 2^{-7} = 1.00\bar{1}000\bar{1}$

CSD number system has advantages over conventional binary system. CSD representation contains fewer non-zero digits therefore there are less partial products in multiplications of the numbers. As an example 15 in binary representation is $(1111)_2$ which contains 4 non-zero digits but in CSD representation, it is $(1000\bar{1})$ which has two non-zero digits. Fig. 1.2 illustrates the comparison between binary and CSD multiplication. It is shown that for the multiplication of 14 and 15 in binary system, 4 additions are needed but in CSD format only 1 subtraction is needed which is the same as addition.

The mathematical operations in digital filters are in the form of multiplication, addition and shift. In the implementation of digital filters, addition and shift operations are cheap. When a digital filter faces a series of input data, multiplication operation forms as Multiple Constant Multiplication (MCM). Multiple Constant Multiplications

Binary Multiplication			CSD Multiplication		
14	00001110		14	000100$\bar{1}$0	
x15	00001111		x15	0001000$\bar{1}$	
	00001110	add		000100$\bar{1}$0	subtract
	00001110	add		000100$\bar{1}$0	shift 3 times add
	00001110	add		00100$\bar{1}\bar{1}$0010 = (210)	
	00001110	add			
	0011010010 = (210)				

Figure 1.2: Comparison of CSD and Binary Multiplication

consume most of the power and implementation cost in digital filters.therefore low implementation cost MCM is always desired. One popular method for reducing the implementation complexity of MCM is to constrain the coefficients to have canonical signed digit (CSD) format with limited number of non-zero digits[1]. [5]. Thereby the heavy MCMs can be replaced by fewer shift and add operation.

By Using the Common Subexpression Elimination (CSE) [37]. [35]. [17], further reduction can also be made in the number of addition in Multiple Constant Multiplication. CSE is the process of finding the common patterns (subexpressions) in a expression and calculate them once and use the result where the subexpression occurs within the expression. Since removing one subexpression may destroy another subexpression, it is critical to find the optimum subexpression for elimination. As an example, in the following expression, choosing the horizontal subexpression (101) will destroy the vertical subexpression (11) and viceversa.

$$c_0 \quad 1010\bar{1}0101$$
$$c_1 \quad 101000101$$

1.3 Summary of Previous Works

Generally there are two approaches for the design of digital filters, direct [19] method and indirect [45] method. In indirect method, first a normalized analog filter using classical approximations methods such as Butterworth, Chebychev, Elliptic, etc [40] is designed, then through some discretization process, the desired digital filter is digitized. Famous discretization procedures are invariant impulse response method, matched z-transformation and bilinear transformation [40].

In direct method [1], [18], by utilizing iterative optimization methods, the desired discrete-time transfer function is calculated. In these methods first a discrete-time transfer function is formed through an initial guess. Then the error function which is based on the desired magnitude and phase response is defined. The optimal answer is obtained by minimizing the error function with respect to the coefficients of the transfer function.

In all of the above-mentioned design techniques, coefficients are calculated with high precision but in actual implementation, either hardware or software, the filter coefficients are stored in finite length registers, so quantization must be applied. Quantization in digital filters may result in unstable filters and/or different response. This has resulted in designs with finite precision coefficients. Four popular iterative optimization techniques are branch and bound optimization [11], discretization and re-optimization [30], simulated annealing [32] and genetic algorithm [44].

Genetic Algorithm (GA) is a stochastic search method that imitates the process of natural selection and evolution. GA possesses many features such as parallelism and multiple objectivity and filters designed by GA have the potential of obtaining near global optimum solution [25]. During the past decade, GA has been successfully used to design digital filters [33], [12], [16]. Also the utilization of CSD coefficients has the advantage of minimizing the implementation cost of digital filters.

In literature there are several proposed techniques for the design of digital filter

using genetic algorithm [27], [26], [18], [1], [23] but none of them has the Common subexpression Elimination as a factor in the fitness function. In this thesis we propose a new algorithm, which include characteristics such as Magnitude Response, Phase Linearity and Common Subexpression Elimination in its objective function. We also have extended the existing algorithm for common subexpression elimination to 3 non-zero digits in vertical position.

1.4 Thesis Objectives

The work presented in this thesis conforms to the following objectives:

1. Study the application of the evolutionary computing such as GA and IP in the design of digital filters.
2. Perform a thorough study on performance of Genetic Algorithm for digital filter design.
3. Develop a new algorithm for the design of high throughput Digital Filter.
4. Extend the Common Subexpression Elimination to 3 non-zero digits.

1.5 Thesis Organization

This thesis is organized as follows: Chapter 2 covers the Genetic Algorithm and Immune Programming. Chapter 3 presents the Common Subexpression Elimination and its extension to 3 non-zero digits for vertical subexpressions. Chapter 4 is the formulation for Digital Filter design using Evolutionary Computing and the proposed algorithm and lastly, Chapter 5 provides concluding remarks.

Chapter 2

Evolutionary Computing

2.1 Introduction

Evolutionary Computing [6] is a group of problem solving methods which are based on biological evolution, such as natural selection and genetic inheritance. From these methods Artificial Neural Networks [39], Genetic Algorithm [50] and Artificial Immune Systems [7] can be named. Each of these methods is inspired from a biological process of human body i.e. Artificial Neural Networks are inspired from the neural system of the body, Genetic Algorithm is inspired from the mechanism of natural evolutionary genetics and Artificial Immune System is inspired from the Immune System of human body. In this chapter two of these paradigms, Genetic Algorithm and Artificial Immune Systems, are discussed.

2.2 Genetic Algorithm

Genetic Algorithm (GA) is a search and optimization algorithm which is based on mechanism of natural evolutionary genetics. It was developed by John Holland [20] in 1975 in his "Adaptation in Natural and Artificial Systems" and was further improved by Goldberg [14], [15] and others [22]. This method works on a population of candidate solutions and tries to find the optimal answer.

GA is based on the mechanism of natural selection and the principal of survival of the fittest. In this algorithm, individuals in a population compete for survival. After each generation the most successful individuals are more likely to survive and the less successful individuals are gradually eliminated.

For solving real-world problems with Genetic Algorithm, the solution to the problem must be expressed as a character string called chromosomes. Also there should be a fitness function to determine the fitness of individuals. In the process of GA, it tries to obtain a better solution from the existing solutions.

As it was mentioned GA is a search-based optimization technique. Search techniques can be divided into three groups, random search, enumerative and calculus based. Fig. 2.1 shows different search techniques.

2.2.1 Genetic Algorithm Cycle

Genetic Algorithm cycle by Goldberg [15] is shown in Fig. 2.2. This algorithm consists of three operations: Reproduction, Crossover and Mutation.

Chromosomes of each population which are the candidate solutions are fed into this cycle. There must be a fitness function defined for each problem to determine the fitness of each solution. Reproduction selects the fitter chromosomes from each population for crossover and mutation. New population is created by each cycle and this process is repeated until the minimum error or maximum number of generations

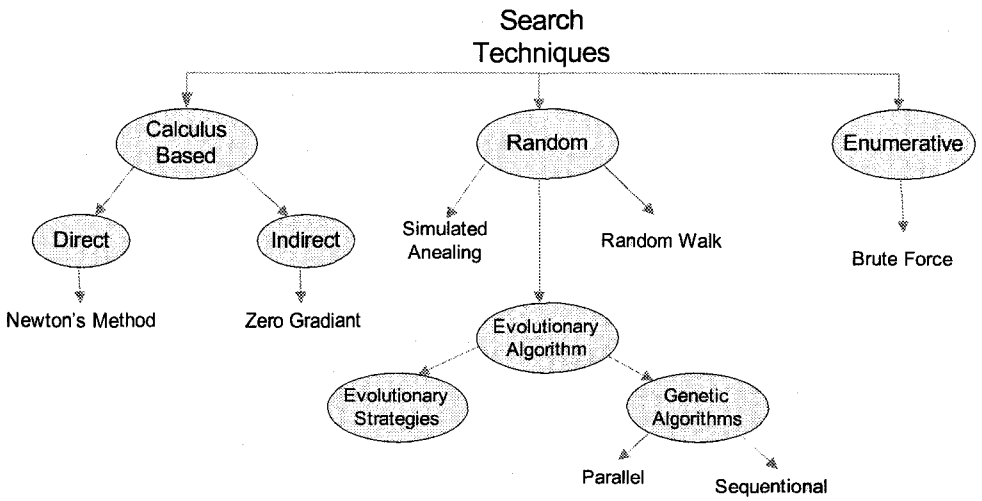


Figure 2.1: Search Techniques.

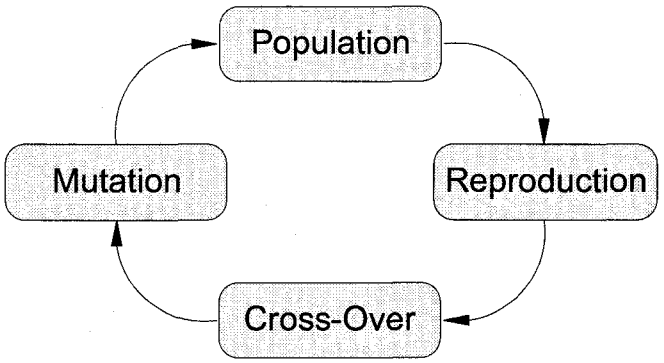


Figure 2.2: GA Cycle.

is achieved. The chromosome that has the highest fitness is the solution to the target problem.

A Genetic Algorithm in its simplest form operates as the following steps:

1. Generating the random initial population.
2. Evaluating the fitness of each chromosome.
3. Selecting a group of chromosomes as parents.

4. Creating new population by crossover and mutation.

5. Replacing the new population according to its fitness.

6. Repeating the loop to reach the target.

In the following sections, each of these steps is discussed in detail.

2.2.1.1 Problem Coding

Encoding scheme [2] is the link between real-world problem and the Genetic Algorithm. The solution of the problem must be encoded to form the chromosomes and each chromosome is a possible solution for the problem. Chromosome consists of string of a characters which can be in binary, integer or any other format.

Encoding of the problem is a crucial issue. Inappropriate coding can intensively limit the searching space which may lead to non-optimal results. The length of the chromosome is also a major factor in Genetic Algorithm. According to the problem in hand, long or short chromosomes may lead to better results [34].

2.2.1.2 Fitness Function

The fitness function is another link of Genetic Algorithm to the real-world problem. Every problem must have an objective function to evaluate the chromosomes. The higher fitness means the better result.

At the start of the GA, after creating the initial random population, the fitness of each chromosome is evaluated and then fed into the GA loop for reproduction.

2.2.1.3 Reproduction

Reproduction is the process of randomly selecting chromosomes with respect to their fitness. Regardless of reproduction mechanism, a chromosome with a higher fitness will have a higher probability of being chosen. This is the simulation of survival-of-the

fittest in natural selection process in which any organism which is more fit will have a better chance to survive in nature.

There are many approaches [49], [51] for reproduction operation such as rank selection, fitness proportionate selection (FPS) and tournament selection. All of these approaches try to give more chances to fitter chromosomes to be selected as parents. A common FPS method is Roulette Wheel Selection. The idea of Roulette Wheel Selection is to divide the wheel to non-uniform slots with respect to chromosomes' fitness. A chromosome with higher fitness will have a bigger slot. When the wheel is spun, the chromosome with the highest fitness has the greatest chance of being chosen. The wheel is spun till the whole population is created. In this method some chromosomes maybe taken more than once. Roulette Wheel executes the following steps:

1. Sum the fitness of all Chromosomes.
2. Generating random number between 0 and total fitness.
3. Choose the chromosome whose fitness added to the fitness of the proceeding chromosomes is less or equal to random number.
4. Repeat the steps till the population size is reached.

Fig. 2.3 is an example of Roulette Wheel Selection. Assume that there are three chromosomes: 01010, 11110 and 11001 with the fitness value of 46%, 35% and 19% of the total fitness. For creating the population, Roulette Wheel is spun three times and in each spin one of the chromosomes is selected. Chromosome 01010 has the 46% of the total fitness, therefore it has the greatest chance of being selected. In the Roulette Wheel Selection, one chromosome can be selected more than once.

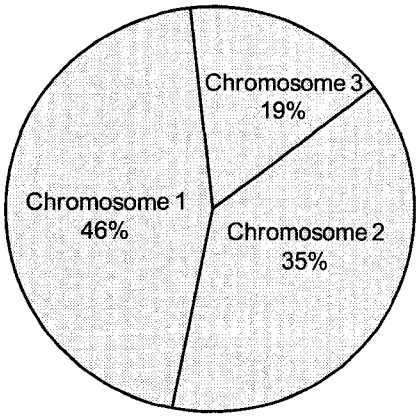


Figure 2.3: Roulette Wheel Selection.

2.2.1.4 Crossover

The selected individuals from the reproduction will be used as parents for crossover. Crossover is the main operator for exploring the searching area. In this operation, according to a probability rate which is Probability of Crossover (P_c), two new chromosomes are generated. As the probability increases, GA can explore more diversity of the solution space but if the P_c is too high, the high fitness chromosomes may destroy [29].

There are variety of crossover operators [21], [43], [38] such as one-point, 2-point and n-point crossover operator. In the following sections, these three approaches will be discussed.

1. 1-point Crossover

   In single point cross-over [10], a cross-over point is chosen at a random position between 1 and the $(string\ length - 1)$. Two new chromosome strings are created by dividing the initial chromosome strings into two sections each at the cross-over point and appending the first half of the first chromosome string to the second half of the second chromosome string and vice versa.

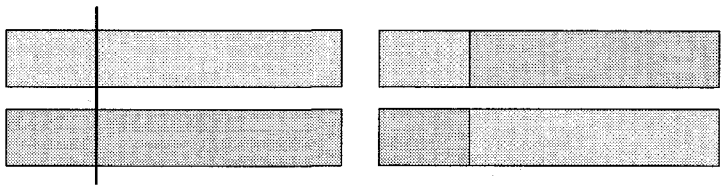


Figure 2.4: 1-point Crossover.

2. 2-point Crossover

   2-point cross-over [4] has chromosomes arranged in loops by joining their ends together. Two cuts are made in the loop and the resulting segments are exchanged. From this it can be seen that 1-point is just a special case of the more general 2-point cross-over where one of the cut points is fixed as falling between the last and first position.

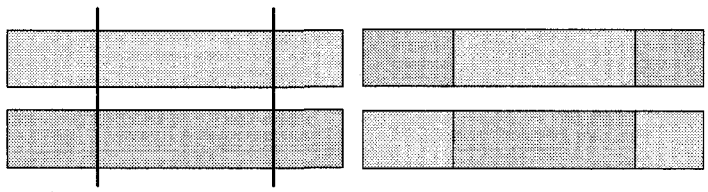


Figure 2.5: 2-point Crossover.

3. N-point Crossover

   Another form of crossover is the n-point crossover [42] where the number of points n varies dynamically with each mating. In this method, a randomly generated crossover mask is used to determine which genes of an offspring come from which parent. Each gene in the first offspring is created by copying the corresponding gene from one or the other parent according to the crossover mask. Where there is a 1 in the mask, the gene is copied from the first parent, and where there is a 0 in the mask, the gene is copied from the second parent. The process is repeated with the parents exchanged to produce the second offspring. A new crossover mask is randomly generated for each pair of parents.

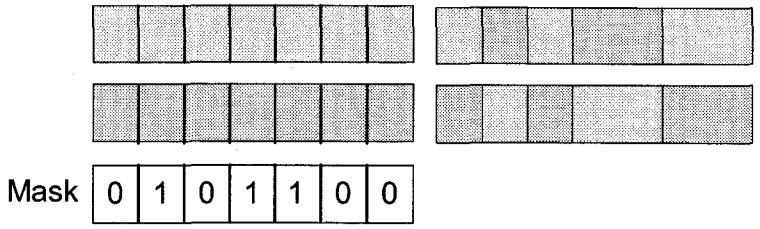


Figure 2.6: Uniform Crossover.

2.2.1.5 Mutation

Normally, mutation [41] occurs with low probability and functions as a background operator. Assume that $A = 10\bar{1}0101\bar{0}\bar{1}$ is the chromosome and **1** is the chosen bit for mutation according to the probability of mutation P_m. As there are totaly three possibilities for strings $(1, \bar{1}, 0)$, **1** must be changed randomly with $\bar{1}$ or 0. If it is changed with 0, the new chromosome will be $A = 10\bar{1}000\bar{1}0\bar{1}$.

2.2.1.6 Replacement

Elitist strategy [24] is applied for the replacement of old generation. After the fitness evaluation, if the maximum fitness of old population is less or equal to the maximum fitness of new population, it is replaced by the new population.

2.2.2 Example

Having introduced the Genetic Algorithm and its operators, now there is an example to demonstrate the GA. In this problem we want to find the maximum of $f(x) = x^2 - x$ where $1 \leq x \leq 5$.

Step1. Variable Coding:

In this problem, variable x is coded as a 5 bit binary number, so a random guess

can be $x = 10110$ which is $x = 22$.

Step2. Initialization:

In this step, a population size of chromosomes are randomly generated to initialize the population. Here we have used 6 as the population size, so the initial population can be:

$a_1 = [10110] = 22$

$a_2 = [00011] = 3$

$a_3 = [11011] = 27$

$a_4 = [00101] = 5$

$a_5 = [11110] = 30$

$a_6 = [01000] = 8$

Step3. Fitness Evaluation:

In this problem we want to find the maximum of $f(x) = x^2 - x$ when $1 \leq x \leq 5$. The function value $f(x)$ is a good candidate for the fitness function. $fitness = f(x) = x^2 - x$.

$fitness(a_1) = f(22) = 462$

$fitness(a_2) = f(3) = 6$

$fitness(a_3) = f(27) = 702$

$fitness(a_4) = f(5) = 20$

$fitness(a_5) = f(30) = 870$

$fitness(a_6) = f(8) = 56$

Step4. Reproduction:

Roulette Wheel Selection is our approach for reproduction operation. Here is the steps:

1. Calculate the total fitness.

$$F = \sum_{i=1}^{6} fitness(a_i) = 2116.$$

2. Generate a random number in the range of [0, 2116]. As an example one can choose the generated number as 2103.

3. Select a chromosome based on the subtotal of chromosomes' fitness and random generated number.

   Table 2.1 shows the subtotal of chromosomes' fitness in this problem.

Table 2.1: Subtotal of Chromosomes' Fitness

Chromosomes	Sum of the Proceeding Fitness
$fitness(a_1) = 462$	462
$fitness(a_2) = 6$	468
$fitness(a_3) = 702$	1170
$fitness(a_4) = 20$	1190
$fitness(a_5) = 870$	2060
$fitness(a_6) = 56$	2116

   According to the Table 2.1 and the number 2103, Chromosome a_5 is selected.

4. Repeat steps 2 and 3 to select the other chromosomes as parents. We assume that the five selected chromosomes are as follow:

   $a'_1 = a_5 = [11110] = 30$

$a'_2 = a_5 = [11110] = 30$

$a'_3 = a_1 = [10110] = 22$

$a'_4 = a_3 = [11011] = 27$

$a'_5 = a_5 = [11110] = 30$

$a'_6 = a_6 = [01000] = 8$

Step5. Crossover:

Creating new population starts by crossover. In this example one point crossover is used and the probability of crossover is 85%. Following is the procedure for crossover:

1. Select two chromosomes randomly. Chromosomes a'_2 and a'_4 are selected.
2. Generate a random number between [0, 1]. Random generated number is 0.543.
3. Since $0.543 < P_c$, crossover operation is performed. Select a crossover point randomly between [1,5]. 3 is selected as crossover point, so a'_2 and a'_3 exchange genes after the crossover point.
4. Repeat steps 1-3 until the population size is complete. In this case 6 chromosomes are created.

   $a''_1 = [11111] = 31$

   $a''_2 = [11010] = 26$

   $a''_3 = [10110] = 22$

   $a''_4 = [11011] = 27$

   $a''_5 = [01111] = 15$

   $a''_6 = [11000] = 24$

Step6. Mutation:

Mutation performs on bits with the probability of 0.05. Each bit of each chromosome is examined with the P_m and if the random generated number is less than P_m, mutation is applied. After mutation the new chromosomes are as follows:

$c_1 = [\mathbf{1}0111] = 23$

$c_2 = [11010] = 26$

$c_3 = [10\mathbf{0}10] = 18$

$c_4 = [11011] = 27$

$c_5 = [\mathbf{1}1111] = 31$

$c_6 = [11000] = 24$

Step7. Fitness Evaluation and Replacement:

$fitness(c_1) = 506$

$fitness(c_2) = 650$

$fitness(c_3) = 306$

$fitness(c_4) = 702$

$fitness(c_5) = 930$

$fitness(c_6) = 552$

After step7, first generation of genetic algorithm is completed. For next generation we continue step 4 to step 7 till reach the maximum number of generations. The best chromosome of the final generation is the optimal answer. In this example chromosome $\{11111\} = 31$ is the maximum of $f(x)$ when $1 \leq x \leq 5$.

2.2.3 GA Analysis

In this section we want to analyze the effect of Genetic Algorithm operators such as Reproduction, Crossover and Mutation. For this purpose the notion of schemata [14] has to be introduced. Schemata is a string composed of the letters of the chromosomes (1, 0 for binary) and * for the don't care position. A schema represents all strings which match all the positions except *. For example for a binary string of length 5, the schemata *011* matches chromosomes 10110, 00110 and 00111.

2.2.3.1 Effect of Reproduction

Assume that the average fitness of the schema h is $m(h,t)$ and $\zeta(h,t)$ be the number of matched chromosomes by the schema h in the current generation. If Roulette Wheel Selection is used as the reproduction operator, the number of chromosomes that match schema h can be estimated in the next generation [46]:

$$\zeta(h,t+1) = \zeta(h,t) \times \frac{m(h,t)}{M(t)} \tag{2.1}$$

which $M(t)$ is the average fitness in the current generation. Let

$$\epsilon = \frac{m(h,t) - M(t)}{M(t)} \tag{2.2}$$

if $\epsilon > 0$, it means that in the current generation, the schema has an above-average fitness. Substituting Equation 2.2 in Equation 2.1, we will have

$$\zeta(h,t+1) = \zeta(h,t) \times (1+\epsilon) \tag{2.3}$$

Starting from $t = 0$, we will obtain the equation

$$\zeta(h,t+1) = \zeta(h,0) \times (1+\epsilon)^t \tag{2.4}$$

Equation 2.4 shows that after reproduction operation, the above-average schema will receive an exponentially increasing number of chromosome in the next generation and the below-average schema will die.

2.2.3.2 Effect of Crossover

In the crossover operation, schema survives only if the crossover point falls outside of the schema i.e if the length of the chromosome is L, and $\delta(h)$ is the order of the schema h, h survives only if the point is outside of its defining length $\delta(h)$.

For the one-point crossover, the probability of distortion of string h is [46]:

$$p_d(h) = \frac{\delta(h)}{L-1} \tag{2.5}$$

therefore the probability of the survival of schema h is [46]:

$$p_s(h) = 1 - \frac{\delta(h)}{L-1} \tag{2.6}$$

Assuming that P_c is the probability of crossover, then the probability of schema survival is:

$$P_{sc}(h) \geq 1 - P_c \frac{\delta(h)}{L-1} \tag{2.7}$$

Equation 2.7 shows that as the order of the schema increases, the probability of the survival of the schema decreases.

2.2.3.3 Effect of Mutation

Mutation works in bit level with a low probability which is called probability of mutation (P_m). The probability of a single bit to survive is $1 - P_m$. A schema survives from distortion only if all the positions in the schema remains unchanged. The formulation for the probability of a schema h surviving the mutation operation is [46]:

$$p_s(h) = (1 - p_m)^{\delta(h)} \tag{2.8}$$

In genetic algorithm $P_m \ll 1$, so Equation 2.8 can be approximated as

$$p_{sm}(h) = 1 - p_m \times \delta(h) \tag{2.9}$$

Equation 2.9 shows that for high order schema, distortion is more probable.

2.2.3.4 Schema Growth Equation

If we combine the effect of reproduction, crossover and mutation, we will obtain the following schema growth equation [46]:

$$\begin{aligned} \zeta(h, t+1) &= \zeta(h,t) \times \frac{m(h,t)}{M(t)} \times p_{sc} \times p_{sm} \\ &\approx \zeta(h,t) \times \frac{m(h,t)}{M(t)} \times [1 - p_c \times \frac{\delta(h)}{L-1} - P_m \times \delta(h)] \end{aligned} \tag{2.10}$$

Equation 2.10 shows that schemata with short, above-average and low-order properties receive exponentially increasing number of representatives in the subsequent generations of a GA, therefore the encoding scheme of the problem must be chosen in a way that these kind of schemata be produced.

2.2.4 Effect of Crossover and Mutation on CSD number

Crossover and Mutation operations may violate the CSD format. These two operations can create invalid coefficients by breaking the two constraints of CSD number system. They can violate the non-zero adjacency constraint by putting two non-zero digits besides each other or increase the number of non-zero digits to more than prespecified limit. Fig. 2.7 shows the effect of crossover on CSD number and Fig. 2.8 shows the effect of mutation on CSD format chromosomes. In chapter 4 we present two different techniques to overcome this problem.

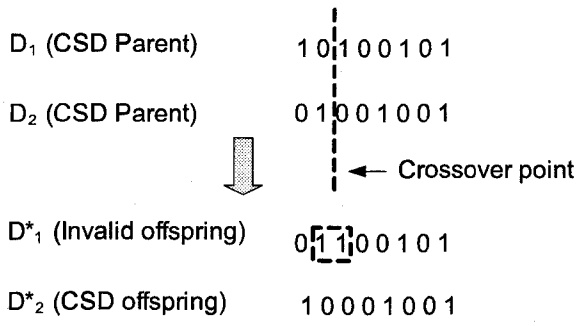


Figure 2.7: Effect of Crossover on CSD coefficients.

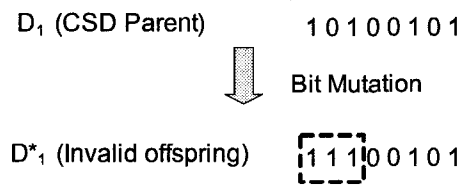


Figure 2.8: Effect of Mutation on CSD coefficients.

2.3 Immune Programming

Artificial Immune System (AIS) [7] is a search and optimization technique which is inspired by the immune system of the body and its principles and mechanisms. Immune programming (IP) [31] is a novel paradigm combining the program-like representation of solutions to problems with the principles and theories of the immune system. Basically, IP is the extension of the Clonal Selection Algorithm in Artificial Immune Systems. IP is not dependent to any domain and it can be applied on a wide variety of problems.

2.3.1 Immune System

The immune system [9] of the body is a natural, rapid and effective defence mechanism for our body to work against infections. Knowing that artificial neural networks are inspired from the nervous system of the body, similar to that, the immune system has led to the emergence of artificial immune systems as a computational intelligence (CI) paradigm.

The immune system consists of two stages of defence which are the innate immune system and the adaptive immune system. The innate immune system works through the cells that are always available. They defend against the wide range of bacteria and they don't need any pre-knowledge of them. The adaptive immune system produces the antibodies only in response to specific infections. These cells have memory therefore they are capable to recognize the same infection when it is presented to the organism again.

2.3.1.1 Pattern Recognition

Lymphocytes which are the components of the white blood cells, are the tools for the pattern recognition in Immune System. Lymphocytes are in two types, B-cells and

T-cells. Both of these two elements have receptors for identifying the antigens but B-cells can recognize the isolated antigen from the outside and T-cells can recognize the antigenic cell complex. Fig. 2.9 shows the difference of these two elements.

The process of pattern recognition is based on matched shapes. When a T-cell or B-cell receptor compliments the antigen, the recognition is performed. There is a concept affinity which is the degree of binding between the receptors and the cell.

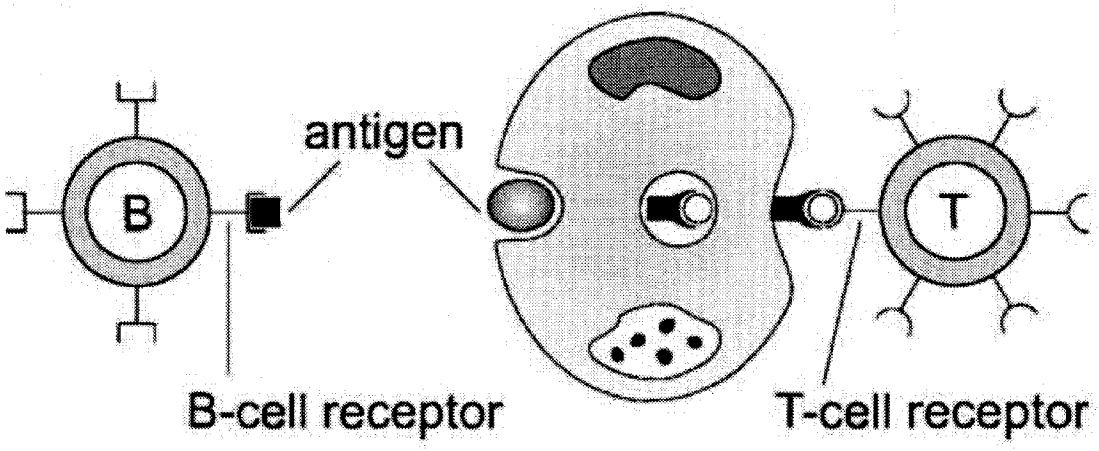


Figure 2.9: Pattern Recognition with B-cells and T-cells.

2.3.1.2 Clonal Selection Algorithm

The Clonal Selection Algorithm (CSA) [8] is the base of Immune Programming. It is the theory that describes the operation of adaptive immune system. According to this theory, only the cells which have the capability of recognizing the antigen can proliferate and the rest are discarded. It works on both B-cells and T-cells with some difference. In CSA B-cells suffer somatic mutation [8] during reproduction but T-cells don't suffer mutation during reproduction.

Fig. 2.10 shows the principle of Clonal Selection Algorithm. In this figure darker circles have the higher fitness and the inner circles in step 3 represent the level of

mutation.

Fig. 2.10 illustrates that after initialization, cells with successful binding are cloned. This will lead to a new population. The new population will be mutated with respect to their fitness to make a new cell with a slight difference. Due to the high mutation rates, this process is usually called hypermutation. This whole process of selection and hypermutation is called clonal selection.

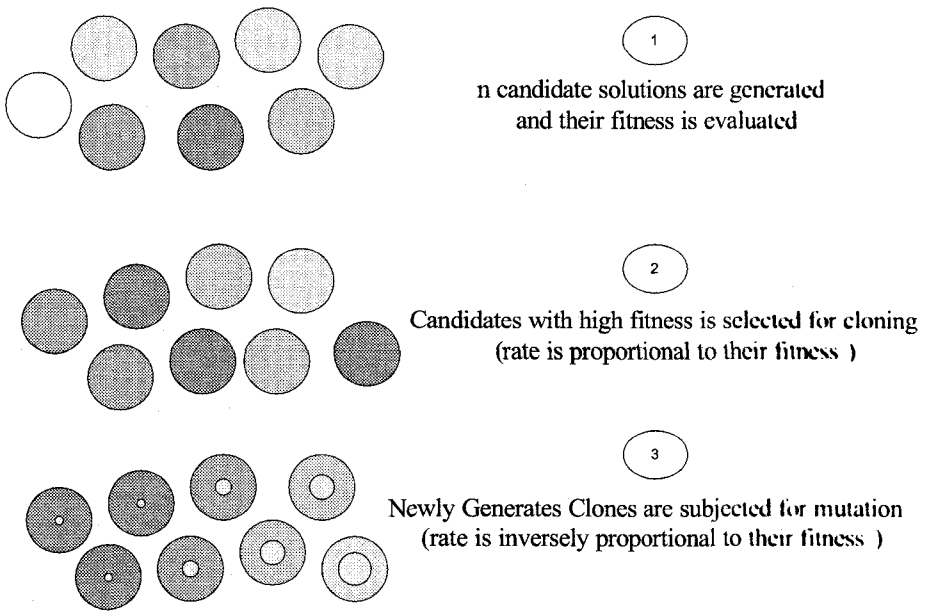


Figure 2.10: Clonal Selection Algorithm.

In summary, the main properties of the clonal selection algorithm are:

- Antibody generation
- Reproduction for high affinity antibodies
- Mutation with respect to affinity

2.3.2 Immune Programming

The IP algorithm is the development of Clonal Selection Algorithm and the replacement of low affinity antibodies. IP consists of the following steps:

1. Initialization
2. Evaluation
3. Replacement
4. Cloning
5. Hypermutation
6. Iteration-repertoire
7. Iteration-algorithm

Step1. Initialization:

In this step, the initial repertoire of antibodies are randomly generated. In IP antibodies can be instruction sets for the problem. Table 2.2 is a sample instruction set which is taken from the [31]. Each instruction is coded and the greatest code number has the value of $r-1$, where r is the number of instructions in the instruction set.

Next step is choosing the length of the program. The length of the program can be variable or fixed. For variable length programs, nop instruction can be used as padding. For the program length hardware constraints should be taken into account as well. As an example according to the Table 2.2, the attribute string of $m =< 1, 4 >$ represents: dup add

Table 2.2: Instruction Set [31]

Instruction	Code	Description
nop	0	No operation
dup	1	Duplicate the top of the stack ($x \rightarrow x\ x$)
swap	2	Swap the top two elements of the stack ($x\ y \rightarrow y\ x$)
mult	3	Multiply the top two elements of the stack ($2\ 4 \rightarrow 8$)
add	4	Add the top two elements of the stack ($2\ 4 \rightarrow 6$)
over	6	Duplicate the second item on the stack ($x\ y \rightarrow y\ x\ y$)

The program length L and the size of instruction size r state the available antibodies for the problem. Here we start with one example to show the different steps of the algorithm.

In this example the length of the program is 5 and the size of the repertoire $n = 100$. Let us consider the following antibodies from the whole repertoire. These antibodies are the possible solution for the problem. The notation AB is reserved for the whole repertoire and the notation Ab_i is reserved for each antibody in the repertoire.

$Ab_1 =< 2, 4, 5, 1, 0 >$;

$Ab_2 =< 1, 3, 0, 5, 0 >$;

$Ab_3 =< 5, 1, 5, 2, 4 >$;

$Ab_4 =< 4, 3, 2, 1, 0 >$.

Step2. Evaluation:

Antibodies are the possible solution for the problem and the antigen is the problem. Antigen can be represented in different formats. In this example the antigen is an arithmetic expression [31].

$Ag = x^2 + x + xy$

Next all of the antibodies are decoded and the corresponding programs are taken out. Then the whole repertoire is compared with the antigen and the affinity is calculated.

Pg_1 = swap add over dup mult;

Pg_2 = dup mult nop add nop;

Pg_3 = over nop over swap add;

Pg_4 = over mult swap dup nop.

To evaluate the affinity of the antibodies numerical argument values must be generated and put into the stack to get the value. These argument can be generated randomly within a prescribed range. In this example argument are restricted to 1-byte integer $[0, ..., 255]$. 4 sets of random arguments are generated as $x = [123, 58, 241, 22]$ and $y = [7, 36, 124, 263]$.

The results of executing the antibodies Ab_i are

Pg_1=[130, 94, 365, 285];

Pg_2=[15136, 3400, 58205, 747];

Pg_3=[130, 94, 365, 285];

Pg_4=[105903, 121104, 7202044, 127292].

Symbol N/A is reserved for the programs that failed to return a result. The antigen A_g yields the values [16113, 5510, 88206, 6292].

Affinity is calculated using the distance between the generated program Pg_i and the antigen A_g, then the three-tiered measure is added. The three tired-measure are:

Excitability. If there is no error after program running, it is assigned a score T_1.

Completeness. If a program execution returns only a single value, score T_2 is added.
Correctness. If the result obtained by the antigen and antibody are identical, score T_3 is added.

The tier scores are defined such that $T_1 < T_2 < T_3$.

This states that correctness is more important than program completeness, which is in turn more important than its executability. The scores can be defined as:

$T_1 = 1$;

$T_2 = c(T_1 + 2)$;

$T_3 = c(T_1 + T_2 + 2)$;

For $c = 5$ the scores will be $T_1 = 1$, $T_2 = 15$ and $T_3 = 90$. The overall affinity of each antibody can be expressed as

$$f_i = \sum_{j=1}^{c} T_1(Pg_i, arg_j) + T_2(Pg_i, arg_j) + T_3(Pg_i, arg_j) \tag{2.11}$$

where arg_j is the j_{th} set of arguments used for program evaluation. In order to normalize the affinities for a given number of independent evaluations f^m is calculated as follows.

$$f^m = c(T_1 + T_2 + T_3) = 530 \tag{2.12}$$

Returning to the running example, the affinities f_i and normalized affinities f_i^N of the generated programs have values $f = [f_1, f_2, f_3, f_4] = [80, 80, 80, 80]$ and $f^N = [0.15, 0.15, 0.15, 0.15]$.

Step3. Replacement:

After evaluating the affinity of the whole repertoire, the algorithm proceeds to create the new repertoire. This process consists of three steps: replacement, cloning

and hypermutation. Cloning and hypermutation is applied to high affinity antibodies and replacement is simply creating new random antibodies. The replaced antibodies will not participate in hypermutation and cloning.

The process of replacement is as follows. First a random number $RAND \in [0, 1]$ is generated and is compared to probability replacement P_r. If the randomly generated number is less than P_r, a new antibody is generated and placed in the new repertoire and the algorithm proceeds to iteration-repertoire process, but if $RAND \geq P_r$ no new antibodies will be generated and the algorithm proceeds to the next step.

As an example, let us consider the new empty repertoire $|AB^{(2)}| = 0$, the probability of replacement is $P_r = 0.65$ and the generated random number is $RAND = 0.235$. Because the random number is less than P_r, a new antibody is generated and placed in the repertoire. After the replacement $|AB^{(2)}| = 1$ and $Ab_1^{(2)} =< 1, 3, 0, 5, 2 >$. The algorithm now proceeds to step 6 (iteration-repertoire).

Step4. Cloning:

While creating a new repertoire, if a new antibody is not generated by replacement, it is cloned from the current repertoire AB. Selecting antibodies for cloning is done sequentially starting from the beginning of the repertoire.

In the cloning step we try to clone the high affinity antibodies rather than low affinity ones. This is accomplished by selecting the high affinity antibodies by a random process. In this process a random number $RAND \in [0, 1]$ is generated and is compared with the relative affinity. If the random number is less than relative affinity $RAND \leq f_i^N$, the antibody will go for cloning or hypermutation according to the P_c and P_m. If the algorithm undergoes the cloning the iteration is done and the algorithm proceeds to step 3, if not, hypermutation is performed. If $RAND \geq f_i^N$, the algorithm return to step 3.

This process is reviewed with an example. Assume the randomly generated number is less than P_r so the algorithm proceeds to cloning. In order to figure out that this antibody is a good candidate for cloning, its relative affinity is compared with a random number. The relative affinity for the selected antibody is $f_1^{(1)N} = 0.63$ and the generated random number is $RAND = 0.235$. In this example $RAND \leq f_1^{(1)N}$, so $Ab_1^{(1)}$ is chosen for cloning or hypermutation. Now another random number is generated $RAND = 0.654$ and the $P_c = 0.7$, because the random number is less than probability of cloning, the $Ab_1^{(1)}$ is cloned and put into the new repertoire.

Step5. Hypermutation:

As it was mentioned in the cloning section, if the antibody is not cloned due to P_c, it is submitted to the hypermutation process. This process works inside the antibody to expand the searching area. Each member of string $m =< m_1, m_2, ..., m_L >\in S^L$ is examined with the probability of hypermutation P_m and a portion of it is replaced with a new randomly-generated value $m_j \in m$.

In the hypermutation process, we try to make less changes on the high affinity antibodies and more changes on the low affinity antibodies, so the probability of mutation must be inversely proportional to affinity. The resulting probability $min[(P_m/f_i^N), 1]$ is used to replace a particular attribute,m_j, of the antibody Ab_i.

Step6. Iteration-Repertoire:

Steps 3 to 5 (replacement, cloning and hypermutation) are repeated until a complete new repertoire, $|AB^{(2)}| = n$, is created. According to the algorithm, there is a pointer in the system to mark which antibodies from the current repertoire are used for cloning or mutation. After all of the repertoire is considered for cloning and

mutation, the pointer is set to 0 again. By this process, high affinity antibodies can be cloned or mutated more than once.

Step7. Iteration-Algorithm:

After the new repertoire is created, the generation counter G is incremented, $G = G + 1$. The algorithm proceeds the iteration steps (evaluation, replacement, cloning, hypermutation, iteration repertoire) until the stop criteria is met. The stop criteria can be a minimum error, constraint on the number of generations or no fitness improvement.

The algorithm is summarized in a block diagram depicted in Fig. 2.11.

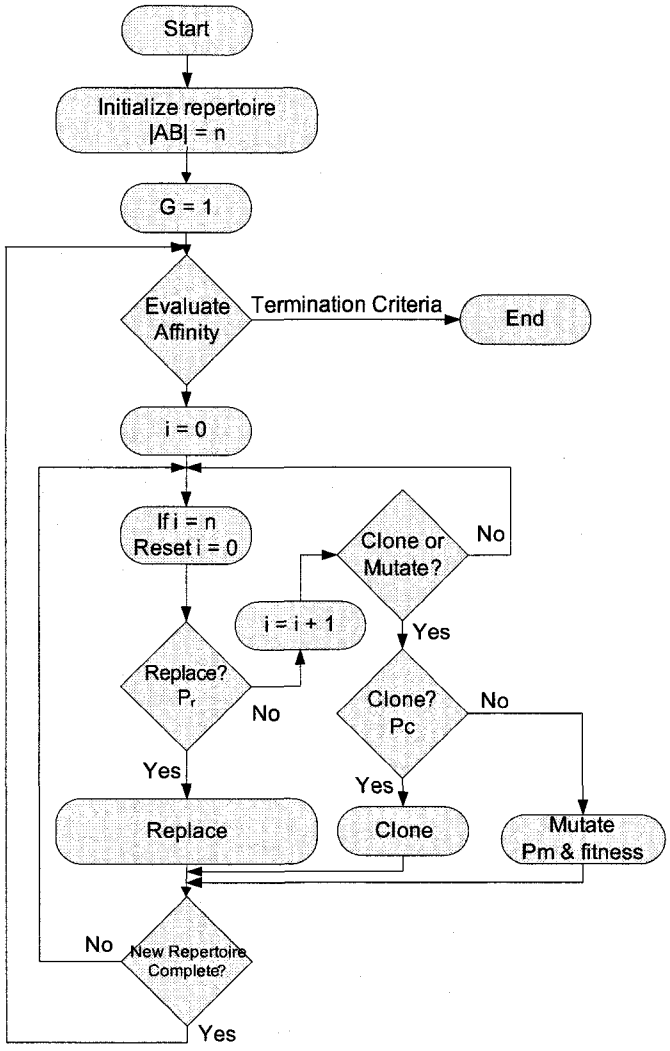


Figure 2.11: Flow chart of the IP algorithm [31].

2.3.3 Example

The same example which was solved by GA is now used to demonstrate the process of IP. The goal of this example is to find the maximum of $f(x) = x^2 - x$ when $1 \leq x \leq 5$. The IP algorithm introduced in previous section is modified for this specific problem.

Before starting the algorithm, the variables should be coded to form the antibodies. In this problem, variable x is coded as a 5 bit binary number, therefore a random antibody can be $Ab = 10110 = 22$.

Step1. Initialization:

The algorithm starts by randomly generating the the initial repertoire $AB^{(1)}$. In this example the size of repertoire is 6. A randomly generated initial population can be:

$$ab_1^{(1)} = [10110] = 22$$
$$ab_2^{(1)} = [00011] = 3$$
$$ab_3^{(1)} = [11011] = 27$$
$$ab_4^{(1)} = [00101] = 5$$
$$ab_5^{(1)} = [11110] = 30$$
$$ab_6^{(1)} = [01000] = 8$$

Step2. Evaluation:

In this problem we want to find the maximum of $f(x) = x^2 - x$ when $1 \leq x \leq 5$. The proposed affinity function in Equation 2.11 is modified for this problem. In this example, antibodies will always yield to an answer, therefore the three-tiered measure is not needed and the affinity function can be the function value $f(x)$, affinity =

$f(x) = x^2 - x$.

affinity$(ab_1^{(1)}) = f(22) = 462$

affinity$(ab_2^{(1)}) = f(3) = 6$

affinity$(ab_3^{(1)}) = f(27) = 702$

affinity$(ab_4^{(1)}) = f(5) = 20$

affinity$(ab_5^{(1)}) = f(30) = 870$

affinity$(ab_6^{(1)}) = f(8) = 56$

Step3. Replacement:

After evaluating the affinity of the whole repertoire, the algorithm proceeds to create the new repertoire. This process starts by replacement.

let us consider the new empty repertoire is $|AB^{(2)}| = 0$, the probability of replacement is $P_r = 0.65$ and the generated random number is $RAND = 0.105$. Because the random number is less than P_r, a new antibody is generated and placed in the repertoire. After the replacement $|AB^{(2)}| = 1$ and $Ab_1^{(2)} = [10001] = 17$. The algorithm now proceeds to step 6 (iteration-repertoire).

Step4. Cloning:

While creating a new repertoire, if a new antibody is not generated by replacement, it is cloned from the current repertoire $AB^{(1)}$.

Assume that the randomly generated number was less than P_r and the algorithm was proceeded to cloning. In order to figure out that this antibody is a good candidate for cloning, its relative affinity is compared with a random number. The relative affinity for the selected antibody is $f_1^{(1)N} = 0.63$ and the generated random number is $RAND = 0.462$. In this example $RAND \leq f_1^{(1)N}$, so $Ab_1^{(1)}$ is chosen for cloning or

hypermutation. Now another random number is generated $RAND = 0.574$ and the $P_c = 0.7$, because the random number is less than probability of cloning, the $Ab_1^{(1)}$ is cloned and put to the new repertoire.

Step5. Hypermutation:

As it was mentioned in the cloning section, if the antibody is not cloned due to P_c, it is submitted to the hypermutation process.

Let us further assume that in the current iteration, replacement does not take place and antibody $Ab_1^{(1)}$ is again considered for cloning due to its high affinity. This time the random number, e.g., RAND = 0.5161, is greater than the probability of cloning, Pc = 0.1, and cloning is not performed. Subsequently, the antibody $Ab_1^{(1)}$ is submitted for hypermutation. $Ab_1^{(1)} = [10110]$ and **1** is chosen for hypermutation. The new generated antibody would be $Ab_1^{(2)} = [10010]$.

Step6. Iteration-Repertoire:

Steps 3 to 5 (replacement, cloning and hypermutation) are repeated until a complete new repertoire, $|AB^{(2)}| = 6$, is created.

$ab_1^{(2)} = [10001] = 17$
$ab_2^{(2)} = [10110] = 22$
$ab_3^{(2)} = [10011] = 19$
$ab_4^{(2)} = [00101] = 5$
$ab_5^{(2)} = [11111] = 31$
$ab_6^{(2)} = [10101] = 21$

Step7. Iteration-Algorithm:

After the new repertoire is created, the generation counter G is incremented, $G = G + 1$ and the stop criteria is checked. In this example. the stop criteria is the number of generations $G = 50$. After 50 generations, the antibody $\{11111\} = 31$ is selected as the maximum of $f(x)$ when $1 \leq x \leq 5$.

2.4 Conclusion

In this section Genetic Algorithm and Immune Programming were introduced and their process were reviewed with an example, also, the effect of their operators, crossover and mutation (hypermutation), on CSD number were discussed. In chapter 4, designing different kinds of digital filters with CSD coefficients using GA and IP is presented.

Chapter 3

Common Subexpression Elimination

3.1 Introduction

High throughput, low power and low implementation cost digital filters are required in DSP applications. Since multipliers in digital filters consume most of the power and implementation cost, high speed and low cost multipliers are required in the implementation of digital filters. As mentioned before, one popular method for reducing the implementation complexity is to constrain the filters to have canonical signed digit (CSD) coefficients with limited number of non-zero digits. Thereby complex multipliers can be replaced by fewer shift and add operations.

By Using the Common Subexpression Elimination (CSE), further reduction can also be made in the number of additions in CSD multiplication [46], [37], [35], [17]. CSE is the process of finding the common patterns (subexpressions) in the expression

and calculate them once and use the results where the subexpression occurs within the expression. In this chapter we have used a graphical method for finding both vertical and horizontal subexpressions for 2 non-zero digits and have extended it to 3 non-zero digits in vertical position.

3.2 Different Subexpressions

Common Subexpressions can be found in horizontal (horizontal subexpression) and vertical (vertical subexpression) position. It can be any bit pattern like 1001, $10\bar{1}$, $10\bar{1}01$. In these patterns we have two or three non-zero digits and some other digits in between.

3.2.1 Horizontal Subexpression Elimination

Within a CSD coefficient, sub-expressions can be found and eliminated horizontally. As an example, we want to calculate Equation 3.1.

$$y = (01010101)_x \tag{3.1}$$

If we express this equation by shift/add method, it changes to Equation 3.2 which needs three additions. The notation $\ll$ is the shift to left operator.

$$y = x + x \ll 2 + x \ll 4 + x \ll 6 \tag{3.2}$$

The subexpression 101 has occurred three times in this expression and two of them can be eliminated. If we rearrange the Equation 3.2, we will have

$$y = x + x \ll 2 + (x + x \ll 2) \ll 4 \tag{3.3}$$

if we take $s = x + x \ll 2$ (101) and rewrite the Equation 3.2, it will be

$$y = s + s \ll 4 \tag{3.4}$$

Equation 3.4 needs one addition and s needs one addition, therefore there is need for 2 additions to calculate y instead of 3 which is needed in Equation 3.2. This represents 33% saving in the number of required additions.

3.2.2 Vertical Subexpression Elimination

Within a CSD coefficients, sub-expressions can also be found and eliminated vertically. As an example, we want to calculate Equation 3.5 [46].

$$\begin{aligned} y &= (10000101)x[0] \\ &+ (10100101)x[-1] \end{aligned} \tag{3.5}$$

If we calculate this equation by shift/add method, it changes to Equation 3.6 which needs 6 additions.

$$\begin{aligned} y = x[0] + x[0] \ll 2 + x[0] \ll 7 + x[-1] \\ +x[-1] \ll 2 + x[-1] \ll 5 + x[-1] \ll 7 \end{aligned} \tag{3.6}$$

The subexpression **11** has occurred twice in this expression and can be eliminated. If we rearrange Equation 3.6, we will have

$$\begin{aligned} y = (x[0] + x[-1]) + (x[0] + x[-1]) \ll 7 \\ +x[0] \ll 2 + x[-1] \ll 2 + x[-1] \ll 5 \end{aligned} \tag{3.7}$$

If we take $s = x[0] + x[-1]$ and rewrite Equation 3.6, it changes to

$$y = s + s \ll 7 + x[0] \ll 2 + x[-1] \ll 2 + x[-1] \ll 5 \tag{3.8}$$

Equation 3.8 needs 4 addition and s needs one addition, therefore there is need for 5 additions to calculate y instead of 6 which is needed in Equation 3.6.

3.3 Design Procedure

Finding and eliminating the optimal subexpressions in the CSD coefficients can make a huge impact on the performance of digital filters. Depending on the problem, sometimes vertical subexpression elimination and sometimes horizontal subexpression elimination yields better performance [3], therefore with no knowledge of the problem it is better to consider both types and try to find the best combination.

There are different approaches for Common Subexpression Elimination in the literature [37], [35], [17]. In this thesis we have used a graphical method for identifying all of the subexpressions and potential elimination paths which can optimally find both vertical and horizontal subexpressions for 2 non-zero digits [46]. It is a two step graphical transformation which transforms the problem to a simple traveling salesman problem [13]. The traveling salesman problem can easily be solved by any optimization method such as Genetic Algorithm. Following are the design step for finding and eliminating the subexpressions with 2 non-zero digits.

3.3.1 Identification Graph

The design procedure starts by creating the identification (ID) graph. This graph contains the information of all horizontal and vertical subexpressions in the coefficients.

$$G_{id} = (V_{id}, E_{id}) \tag{3.9}$$

For creating the ID graph, first the coefficients are stacked vertically to facilitate the identification of horizontal and vertical subexpressions. Next the graph of vertices is created according to:

$$V_{id} = \{\text{non-zero digits in all coefficients}\} \tag{3.10}$$

Then the partial identification graph $G'_{id} = (V_{id}, E'_{id})$ is created by adding E'_{id} which is all of the possible vertical and horizontal subexpressions and defined as $E'_{id} = E_h + E_v$.

$$E_h = (V_a, V_b) \; \forall \; V_a, V_b \; \{\text{within the same coefficient}\} \tag{3.11}$$

and

$$E_v = (V_a, V_b) \; \forall \; V_a, V_b \; \{\text{within the same vertical bit location}\} \tag{3.12}$$

Thus, E_h and E_v are the edges for fully connected sub-graphs in both horizontal and vertical dimensions.

For finding the subexpression that are good candidates for elimination, first we should find the properties of all edges. The properties of edges are type (horizontal and vertical), polarity and length. Polarity shows if two non-zero digits have the same sign or opposite sign. For example, in 1001 and $\bar{1}00\bar{1}$ the multiplication of two non-zero digits ((1,1) and $(\bar{1}, \bar{1})$) yields to positive polarity but in $100\bar{1}$ the multiplication of non-zero digits yields to negative polarity. For horizontal edges length is defined as the number of digits between two non-zeroes and for vertical edges is the distance between the coefficient.

The next step after finding all of the possible edges is to find the edges without common properties (Unique Edges) and remove them from the edge list to create the final ID graph. The final edges are

$$E_{id} = E'_{id} - E_{unique} \tag{3.13}$$

Where E_{unique}= { edges with unique properties }. Following is an example for the design procedure.

Assume that a filter has the CSD coefficients

$$c_0 = 10100\bar{1}01, c_1 = 10100101 \tag{3.14}$$

As it was mentioned, the first step is to stack coefficients vertically. Table 3.1 is the stacked coefficients.

Table 3.1: Coefficient Stacking

c_0	$10100\bar{1}01$
c_1	10100101

The next step is to create the graph of vertices. $V_{id} = \{V_1 ... V_8\}$ represents the all non-zero digits of the coefficients. Fig. 3.1 shows the graph of vertices and table 3.2 is the vertices and their properties.

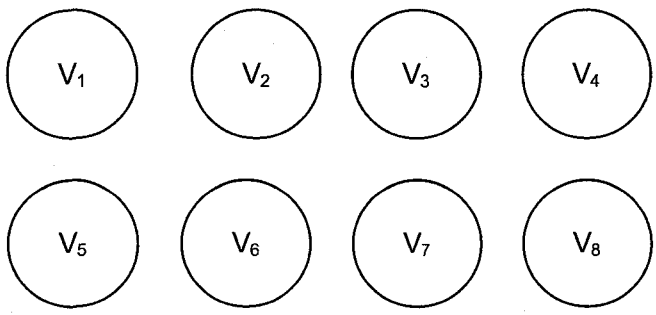


Figure 3.1: Graph of Vertices.

To create the partial ID graph G'_{id}, which represents all of the vertical and horizontal subexpressions, edges are added to the graph of vertices.

Table 3.2: Coefficient Stacking

Vertex	Digit Polarity	Coefficients	Digit Position
V_1	+1	0	8
V_2	+1	0	6
V_3	-1	0	3
V_4	+1	0	1
V_5	+1	1	8
V_6	+1	1	6
V_7	+1	1	3
V_8	+1	1	1

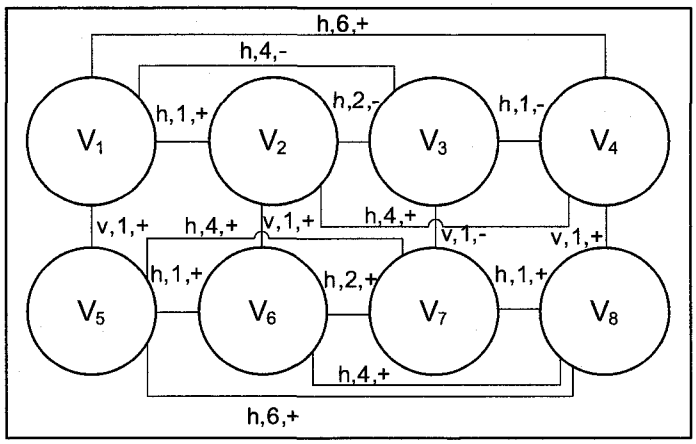


Figure 3.2: Partial Identification Graph G'_{id}.

Table 3.3 shows all the possible edges and their properties. To complete the design and make the completed ID graph G_{id}, the edges that do not share the common properties should be taken out from the partial ID graph. Fig. 3.3 shows the G_{id} which only includes common vertical and horizontal subexpressions.

Table 3.3: Edge List E'_{id} for 2 non-zero digits $\{a = 1, 2, ..., 8,\ b = 1, 2, ..., 8\}$

Edge	(V_a, V_b)	Type	Polarity	Length	Vertical
1	(1,2)	h	+	1	—
2	(1,3)	h	-	4	—
3	(1,4)	h	+	6	—
4	(2,3)	h	-	2	—
5	(2,4)	h	+	4	—
6	(3,4)	h	-	1	—
7	(1,5)	v	+	1	(c_0, c_1)
8	(2,6)	v	+	1	(c_0, c_1)
9	(3,7)	v	-	1	(c_0, c_1)
10	(4,8)	v	+	1	(c_0, c_1)
13	(5,6)	h	+	1	—
14	(5,7)	h	+	4	—
15	(5,8)	h	+	6	—
16	(6,7)	h	+	2	—
17	(6,8)	h	+	4	—
18	(7,8)	h	+	1	—

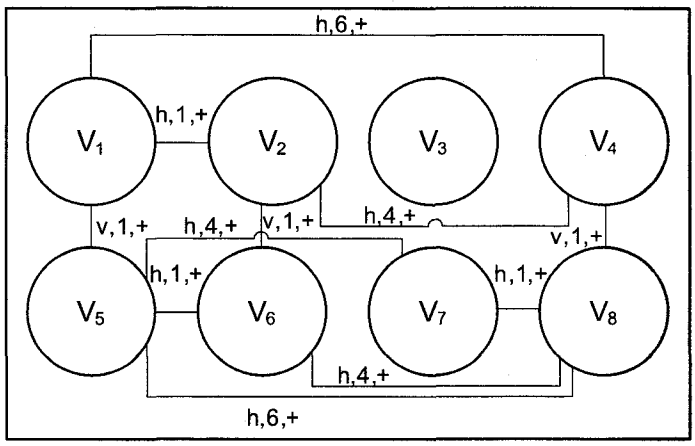


Figure 3.3: Competed Identification Graph G_{id}.

3.4 Common vertical subexpression for 3 non-zero digits

In the previous sections, finding and eliminating common subexpressions for 2 non-zero digits in both vertical and horizontal positions were presented. In this section, we propose common subexpression elimination for 3 non-zero digits in vertical position. Since in digital filter application with CSD number system there are at most 3 to 6 non-zero digits in each coefficient, we are not looking for 3 non-zero digits subexpressions in horizontal position.

For 3 non-zero digits vertical common subexpression elimination, 2 length properties and 2 polarity properties for each edge is added to the edge list. Finding common subexpression for both 2 and 3 non-zero digits is described with an example as follow.

Assume that a filter has the CSD coefficients $c_0 = 10100\bar{1}01$, $c_1 = 10100101$ and $c_2 = 10000001$. The first step is to stack the coefficients vertically (Table 3.4). Then the graph of vertices (Fig. 3.4) is created which represents all of the non-zero digits in the coefficients. Table 3.5 presents the vertices and their properties.

Table 3.4: Coefficient Stacking

c_0	$10100\bar{1}01$
c_1	10100101
c_2	10001001

To create the partial ID graph, E'_{id} which represents all of the vertical and horizontal subexpressions edges, is added to the graph of vertices. Table 3.6 shows the edges and their properties for 2 non-zero digit subexpressions and Table 3.7 presents for 3 non-zero digit vertical subexpressions. Fig. 3.5 is the partial ID graph and dotted lines represent the 3 non-zero digits vertical subexpressions.

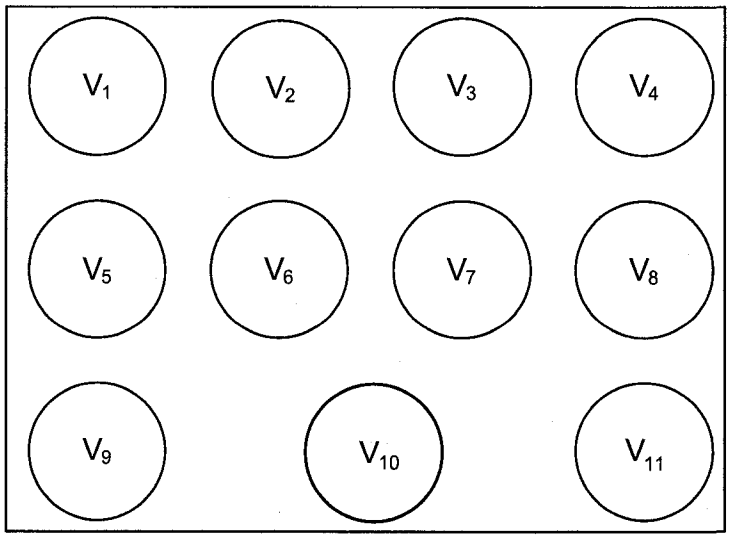


Figure 3.4: Graph of Vertices.

Table 3.5: Vertices and their properties

Vertex	Digit Polarity	Coefficients	Digit Position
V_1	+1	0	8
V_2	+1	0	6
V_3	-1	0	3
V_4	+1	0	1
V_5	+1	1	8
V_6	+1	1	6
V_7	+1	1	3
V_8	+1	1	1
V_9	+1	2	1
V_{10}	+1	2	4
V_{11}	+1	2	8

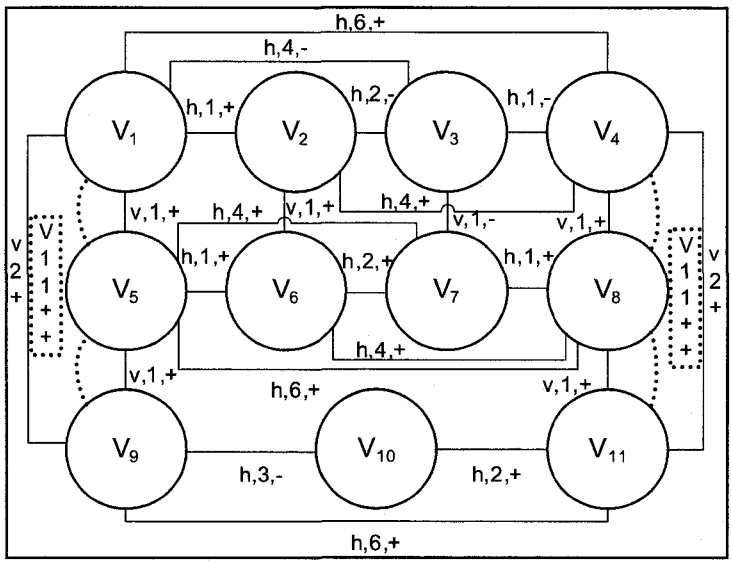


Figure 3.5: Partial Identification Graph G'_{id}.

Table 3.6: Edge List E'_{id} for 2 non-zero digits

Edge	(V_a, V_b)	Type	Polarity	Length	Vertical
1	(1,2)	h	+	1	—
2	(1,3)	h	-	4	—
3	(1,4)	h	+	6	—
4	(2,3)	h	-	2	—
5	(2,4)	h	+	4	—
6	(3,4)	h	-	1	—
7	(1,5)	v	+	1	(c_0, c_1)
8	(1,9)	v	+	2	(c_0, c_2)
9	(2,6)	v	+	1	(c_0, c_1)
10	(3,7)	v	-	1	(c_0, c_1)
11	(4,8)	v	+	1	(c_0, c_1)
12	(4,11)	v	+	2	(c_0, c_2)
13	(5,6)	h	+	1	—
14	(5,7)	h	+	4	—
15	(5,8)	h	+	6	—
16	(6,7)	h	+	2	—
17	(6,8)	h	+	4	—
18	(7,8)	h	+	1	—
19	(5,9)	v	+	0	(c_1, c_2)
20	(8,11)	v	+	1	(c_1, c_2)
21	(9,10)	h	+	3	—
22	(9,11)	h	+	6	—
23	(10,11)	h	+	2	—

Table 3.7: Edge List E'_{id} for 3 vertical non-zero digits

Edge	(V_a, V_b, V_c)	Type	Pol.$_1$	Pol.$_2$	Len.$_1$	Len.$_2$
1	(1,5,9)	v	+	+	1	1
2	(4,8,11)	v	+	+	1	1

To complete the design and make the completed ID graph G_{id}, the edges that do not share the common properties should be taken out from the partial ID graph. Fig. 3.6 shows the G_{id} which only includes common vertical and horizontal subexpressions.

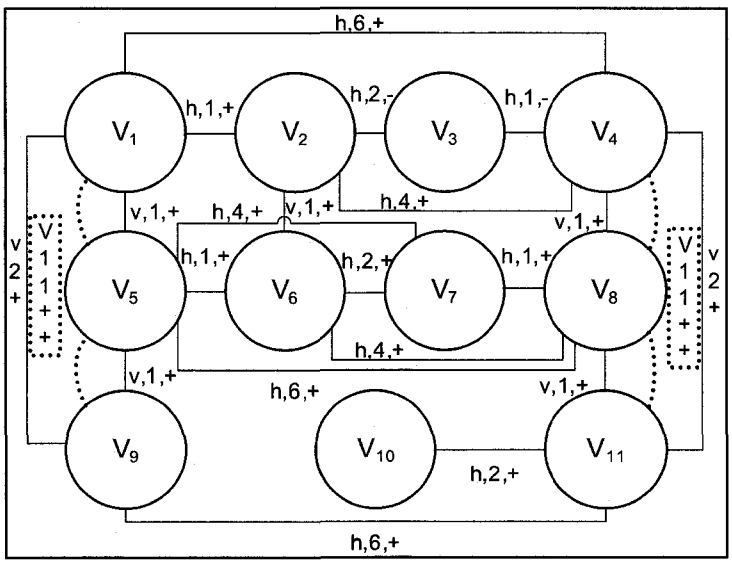


Figure 3.6: Competed Identification Graph G_{id}.

3.4.1 Search Graph

Now that we have the ID graph completed, we can create the search graph. Each edge of the ID graph is a vertex of the search graph $G_s = (V_s, E_s)$.

Since each vertex in the search graph is a subexpression, a hamiltonian Walk

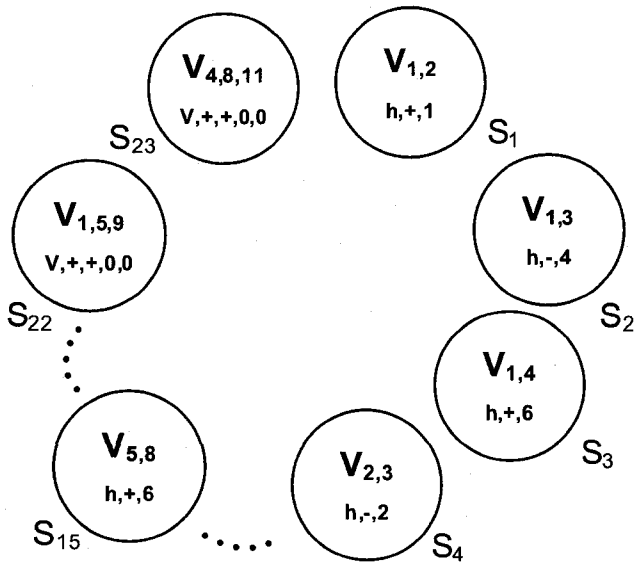


Figure 3.7: Search Graph G_s.

through the vertices can lead to one elimination scheme. Now with a search and optimization algorithm, the best possible walk with the highest number of eliminations can be found.

This problem is quite similar to the Traveling Salesman Problem (TSP) where the vertices are the cities and the edges are the distance between the cities. The objective function is now to find the hamiltonian walk which leads to the smallest sum of edge distances. The difference here is that in the subexpression elimination problem the elimination can be done only if two subexpressions with common properties traversed and that may not be happened all the time.

In the hamiltonian walk an elimination can happen only if the non-zero digits of the edges have not been included in previous eliminations, therefore there is a possibility of finding some paths with common properties but with unavailable edges which will not lead to elimination.

Fig. 3.7 illustrates the search graph G_s.

3.4.2 Example Walk Through the Search Graph

As an example, a sample Hamiltonian Walk can be $S_4, S_3, S_{15}, S_{23}, ..., S_2$. In this example walking from S_4 to S_3 will not lead to elimination because they don't possess same properties and are not common subexpression. The next path is S_3 to S_{15}. S_3 has the properties of $(h, +, 6)$ and S_{15} has the properties of $(h, +, 6)$, so they are a good candidate for elimination.

As we discussed earlier, to eliminate the candidate subexpressions, availability check should be performed. In this elimination V_1 and V_4 from S_3 and V_5 and V_8 from S_{15} are required. Since the (S_3, S_{15}) elimination is the first elimination, all of the vertices are available. We can create a table (Availability Table) and mark the unavailable vertices after every elimination. By using this table available vertices for the next elimination can be found easily.

Table 3.8 illustrates the availability table for the ID graph. According to this table after elimination of S_3 and S_{15}, other possible eliminations such as (S_7, S_9) can not occur.

3.5 Elimination using GA

Finding an optimal Hamiltonian Walk that leads to the maximum number of eliminations can be done by any search and optimization technique. One possible method is Genetic Algorithm (GA). In this problem, chromosomes are the possible walks through the search graph.

For the fitness function, we should count the number of eliminations in each Hamiltonian Walk. At the start of the GA, all of the vertices in the availability table are marked as available. This shows that all of the non-zero digits are available. Traversing the search graph G_s by Hamiltonian Walk which is defined by the chromosome will lead to the first elimination. After each elimination the availability table is modified

Table 3.8: ID Graph Vertex Availability Table after (S_3, S_{15}) Elimination

V_1	unavailable
V_2	Available
V_3	Available
V_4	unavailable
V_5	unavailable
V_6	Available
V_7	Available
V_8	unavailable
V_9	Available
V_{10}	Available
V_{11}	Available

and those vertices which were included in the elimination are marked as unavailable. For each elimination the occurrence count will be incremented.

The fitness value is then determined by summing the n occurrence count (OC) values as shown below:

$$fitness = \sum_{i=1}^{n} (OC_i - 1, \text{ if } OC_i > 1 \text{ else } 0) \tag{3.15}$$

After that the GA is complete, we will find the fittest chromosome, which shows the maximum number of eliminations that can be achieved.

3.6 Experimental Results

This method is applied to FIR filters with the order of 19, 21 and 23 with maximum number of 3, 4 and 5 allowable non-zero digits in each coefficient. Our simulations show that for 2 non-zero digit subexpression, we will gain approximately 25% reduc-

tion in number of additions and by adding 3 non-zero digits in vertical subexpressions, we will obtain 32% of reduction.

Table 3.9: CSE on FIR filter

Filter Order	Non-zero Digits	Original Addition	No. of CSE (2 non-zero)	No. of CSE (3 non-zero)	CSE percent (2 non-zero)	CSE percent (3 non-zero)
19	3	50	13	17	26.00%	34.00%
19	4	56	13	19	23.21%	33.93%
19	5	56	16	19	28.57%	33.93%
21	3	51	13	19	25.49%	37.25%
21	4	67	18	22	26.87%	32.84%
21	5	76	19	26	25.00%	34.21%
23	3	49	11	13	22.45%	26.53%
23	4	63	17	18	26.98%	28.57%
23	5	74	21	23	28.38%	31.08%

3.7 Conclusion

In this chapter a graphical method for common subexpression elimination was presented. The original method was for 2 non-zero bits in vertical and horizontal position. We expanded this method for 3 non-zero digits in vertical position which led to more efficient multiplication. Simulation results for 3 non-zero digits show a computational saving of up to 31% which is 6% more than that of a 2 non-zero digits elimination.

Chapter 4

Formulation of Digital Filter Design using Evolutionary Computing

4.1 Introduction

In this chapter we utilize two evolutionary computing techniques namely Genetic Algorithm and Immune Programming as optimization methods to design Digital Filters. We start by presenting different techniques for the design of digital filters using GA, then we perform a thorough performance analysis of Genetic Algorithm. Furthermore, design of digital filters using Immune Programming is illustrated and at the end a new algorithm for the design of digital filters using GA is presented.

4.2 Design of digital filters using GA

In this section design of FIR and IIR digital filters and also FIR and IIR QMF banks with CSD coefficients is presented. Linear phase characteristics in the pass band for IIR filters is also considered. This section starts by the general design flow for digital filters followed by two different techniques for CSD coding schemes and comparison between them. At the end several performance analysis on GA is conducted.

4.2.1 General Design Flow

Fig. 4.1 shows the proposed GA-based design flow for digital filters. This design methodology is for FIR and IIR digital filters and also FIR and IIR Quadrature Mirror Filter banks. For the case of QMF banks, as we are interested in linear time invariant filters with no aliasing effect, the design will be restricted to design of one of the synthesis or analysis filters as a FIR or IIR filter (Equation 1.7). Therefore by designing a simple FIR or IIR filter we can have the QMF bank designed.

4.2.1.1 Initialization

The design starts by generating random numbers in CSD format as the coefficients of the digital filter for initial population. Each chromosome is constructed by concatenating all the coefficients in the transfer function, thus for an N^{th} order IIR or FIR filter, chromosome is presented as:

$$\begin{aligned} \text{IIR} &: (a_0, a_1, a_2, ..., a_n, b_0, b_1, ..., b_n) \\ \text{FIR} &: (a_0, a_1, a_2, ..., a_n) \end{aligned} \tag{4.1}$$

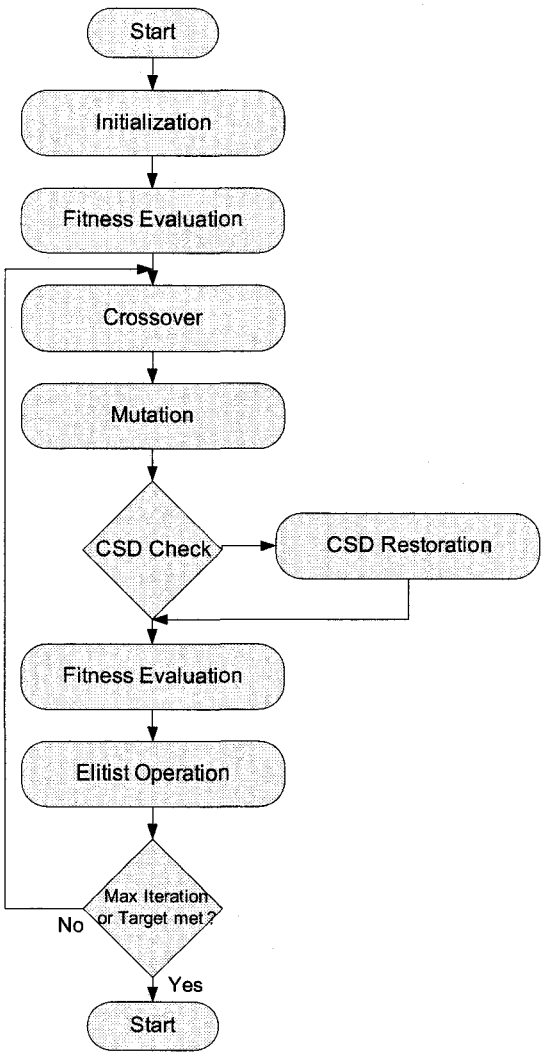


Figure 4.1: GA Design Flow.

4.2.1.2 Fitness evaluation

In this step, the desired filter specification is defined. depending on the type of the filter, different fitness functions are considered. For the FIR filters only the error of the magnitude response is considered which is:

$$E_{mag}(jw_m) = |H_1(exp(jw_mT))| - |H_D(exp(jw_mT))| \quad (4.2)$$

$$E_{FIR} = \sum_{m \in I_{ps}} E^2_{mag}(jw_m) \quad (4.3)$$

$$fitness = \frac{1}{E_{FIR}} \quad (4.4)$$

For IIR filters if linear phase characteristics in the pass-band is desired, the group delay should be taken into account too. $E_\tau(jw_m)$ is the group delay error and usually τ_1 is the filter order [40].

$$E_{mag}(jw_m) = |H_1(exp(jw_mT))| - |H_D(exp(jw_mT))| \quad (4.5)$$

$$E_\tau(jw_m) = \tau_1 T - \tau_D(jw_mT) \quad (4.6)$$

$$E_{IIR} = \alpha \sum_{m \in I_{ps}} E^2_{mag}(jw_m) + (1-\alpha) \sum_{m \in I_p} E^2_\tau(jw_{m1}) \quad (4.7)$$

$$fitness = \frac{1}{E_{IIR}} \quad (4.8)$$

In Equation 4.7 and 4.3, I_{ps} is set of frequency points along w axis in the pass-band and stop-band and I_p is set of frequency points in the pass-band only. α is the weighting factor to emphasize the magnitude or group delay characteristics.

For the IIR filters, stability check is performed and a penalty factor is introduced for unstable filters [46]. By incorporating this penalty factor to the fitness function, unstable filters have much less chance to be chosen as the optimal answer.

$$fitness = \frac{fitness}{penalty} \quad (4.9)$$

The optimum penalty factor was determined empirically through a series of test designs over 100 filters and 300 generations for each design[46]. For any unstable filter occurring in any generation, its fitness penalized by the penalty factor, and with the penalty factor 9 no unstable filter were produced.

4.2.1.3 Reproduction

Roulette Wheel Selection is used as the reproduction operator. The idea behind this technique is that each individual chromosome is given a chance to become a parent proportional to its fitness.

4.2.1.4 Crossover

Crossover is the first step for exploring the search area and creating the new population. 1-point, 2-point and uniform are the most popular techniques for cross-over. A comparison among different crossover techniques is given in section 4.2.4.2.

4.2.1.5 Mutation

Normally, mutation occurs with low probability and functions as a background operator. As an example, on each chromosome of population, with a low probability, each digit of the CSD number is changed with a new one. For example, if our chromosome is $100\bar{1}01001001\bar{1}$ and **1** is the candidate for mutation, according to the probability of mutation, it is changed with 0 or $\bar{1}$.

4.2.1.6 CSD Check

Considering the nature of mutation and cross-over operations, they may violate the CSD format of the coefficients (Fig. 2.7and 2.8), as a result, CSD restoration should be performed in each loop. Two different coding techniques for overcoming this problem is presented in section 4.2.2.

4.2.1.7 Replacement Strategy

Elitist strategy is applied for the replacement of old generation. After the fitness evaluation, if the maximum fitness of old population is less or equal to the maximum fitness of new population, it is replaced by the new population.

4.2.2 Different Coding Techniques

Having introduced the general design flow for the design of digital filters and modifications for the different kinds of filters, now we present two different coding techniques for solving the effect of crossover and mutation on CSD numbers.

4.2.2.1 Technique 1 (Ternary Coding)

This technique [28] starts by initialization which is creating the chromosomes for the first population with random CSD numbers. These chromosomes are fed into the genetic algorithm. GA selects the higher fitness chromosomes by reproduction operation and then performs crossover and mutation to create the new population. As mentioned before, because of the nature of crossover and mutation operations, the CSD format of the chromosomes may be violated. Violated chromosome may have two adjacent non-zero digits or more non-zero digits than the specified limit.

By using this technique if any violated coefficient from the CSD format is found, first the coefficient will be converted to its decimal number and then the decimal number will be converted to its nearest CSD number. After the restoration part, the fitness of the new generation is calculated and the replacement is performed.

As an example *dec* is the decimal representation of the violated coefficient. We want to convert it to a $M = 16$ bit CSD number with the maximum of $L = 4$ non-zero digits, represented by $\{d_0, d_1, d_2, ..., d_{15}\}$. Following is the pseudo-code for this restoration technique.

Pseudo Code of the Restoration Technique

void DecimalToCSD () {
if (*dec* is out of range AND $dec > 0$) $\{d_0 = d_2 = d_4 = d_6 = 1$, conversion completed$\}$
elseif (*dec* is out-of-range AND $dec < 0$)

$\{d_0 = d_2 = d_4 = d_6 = -1$, conversion completed$\}$

/*start from d_0 until d_{15}, find whether a -1, 0 or 1 should be put on d_i*/

else {

for (let variable i changes from 0 to 15, the step is 1) {

if $(counter \geq 4)$ the conversion is completed

elseif $(dec - dec_{d_0-d_i} = 0)$ { $d_i = 1$, conversion completed }

elseif $(dec + dec_{d_0-d_i} = 0)$ { $d_i = -1$, conversion completed }

elseif $(\mid dec - dec_{d_0-d_i} \mid \leq \max dec_{(d_{i+2}-d_{15})})\{d_i = 1, d_{i+1} = 0$, counter+1$\}$

elseif $(\mid dec + dec_{d_0-d_i} \mid \leq \max dec_{(d_{i+2}-d_{15})})\{d_i = -1, d_{i+1} = 0$, counter+1$\}$

$else\{d_i = 0\}$

}

}

}

4.2.2.2 Technique 2 (New Coding Scheme)

In this technique [46], CSD numbers are presented through a new coding scheme. This scheme generates a string of symbols to indicate the position and sign of non-zero digits in a CSD format.

Table 4.1 shows the proposed coding representation for a 6-bit CSD number. Suppose a CSD coefficient is represented by the three digit partial chromosome 19B. Three digits shows that the number has 3 non-zero digits. From Table 4.1 the 1 would signify a +1 in position 2, the 9 would signify a -1 in position 4, and the B would signify a -1 in position 6. Therefore the CSD coefficient in question would be 27.

Under this coding scheme, CSD coefficients with two identical symbols, simply have two non-zeros occupying the same position, so the second non-zero is just ignored since a non-zero already exists at that location. The canonical adjacency constraint is also addressed using the same conflict resolution method. If a partial chromosome

digit indicates that a non-zero digit should be positioned adjacent to another one, the second one is simply ignored.

Table 4.1: Conversion Table For CSD Length = 6

Symbol	0	1	2	3	4	5	6	7	8	9	A	B
Digit Sign	+	+	+	+	+	+	-	-	-	-	-	-
Digit Position	1	2	3	4	5	6	1	2	3	4	5	6

4.2.3 Experimental Results and the Comparison of the Two Coding Scheme

In this section, examples for the design of digital filters using Genetic Algorithm are given. Through these examples different FIR and IIR filters with different coding schemes and phase characteristics are designed. Furthermore, comparison of the two coding schemes for the CSD number system is presented. In following examples $w_s = 2\pi$.

4.2.3.1 FIR filters with ternary CSD coding

In this example, a 19^{th} order, low-pass FIR filter with turnery CSD coefficients and following specification is designed.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.10}$$

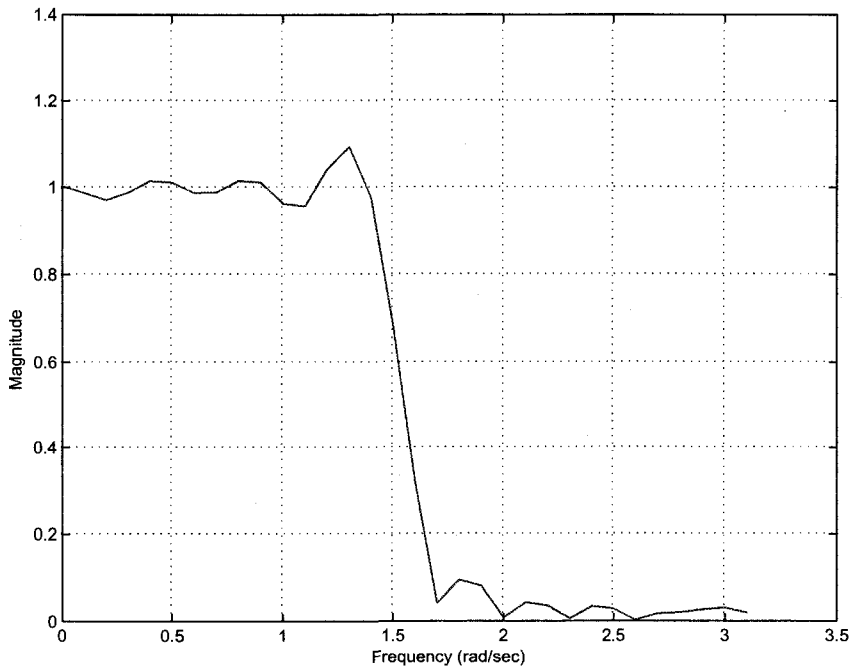


Figure 4.2: 19^{th} order, low-pass FIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Ternary Coding)

Table 4.2: CSD Coefficients of the 19^{th} order low-pass FIR filter with ternary coding

Coefficients	Binary Coefficients
a_0	$2^{-4} - 2^{-7} + 2^{-10}$
a_1	$2^{-7} - 2^{-9} - 2^{-11} - 2^{-13}$
a_2	$2^{-8} + 2^{-10} + 2^{-12} + 2^{-14}$
a_3	$-2^{-5} - 2^{-7} + 2^{-10} - 2^{-13}$
a_4	$2^{-2} - 2^{-7} - 2^{-11} + 2^{-14}$
a_5	$2^{-6} + 2^{-8} + 2^{-15}$
a_6	$2^{-4} + 2^{-6} - 2^{-12} - 2^{-14}$
a_7	$-2^{-7} - 2^{-9} - 2^{-11} - 2^{-13}$
a_8	$2^{-6} + 2^{-9} - 2^{-12} - 2^{-14}$
a_9	$2^{-2} + 2^{-6} - 2^{-8} - 2^{-15}$
a_{10}	$2^{-1} - 2^{-3} - 2^{-9} - 2^{-11}$
a_{11}	$2^{-6} - 2^{-8} + 2^{-11} - 2^{-15}$
a_{12}	$2^{-8} - 2^{-10} + 2^{-12} - 2^{-15}$
a_{13}	$2^{-5} + 2^{-8} - 2^{-10}$
a_{14}	$2^{-2} - 2^{-4} - 2^{-10} - 2^{-14}$
a_{15}	$2^{-7} - 2^{-9} - 2^{-11} - 2^{-13}$
a_{16}	$2^{-3} + 2^{-6} + 2^{-9} - 2^{-14}$
a_{17}	$2^{-4} - 2^{-7} + 2^{-11}$
a_{18}	$-2^{-8} + 2^{-11} + 2^{-12} - 2^{-14}$
a_{19}	$2^{-3} + 2^{-5} - 2^{-11} - 2^{-15}$

4.2.3.2 IIR filters with ternary CSD coding

In this example, a 5^{th} order, low-pass IIR filter with ternary CSD coefficients and following specification is designed.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.11}$$

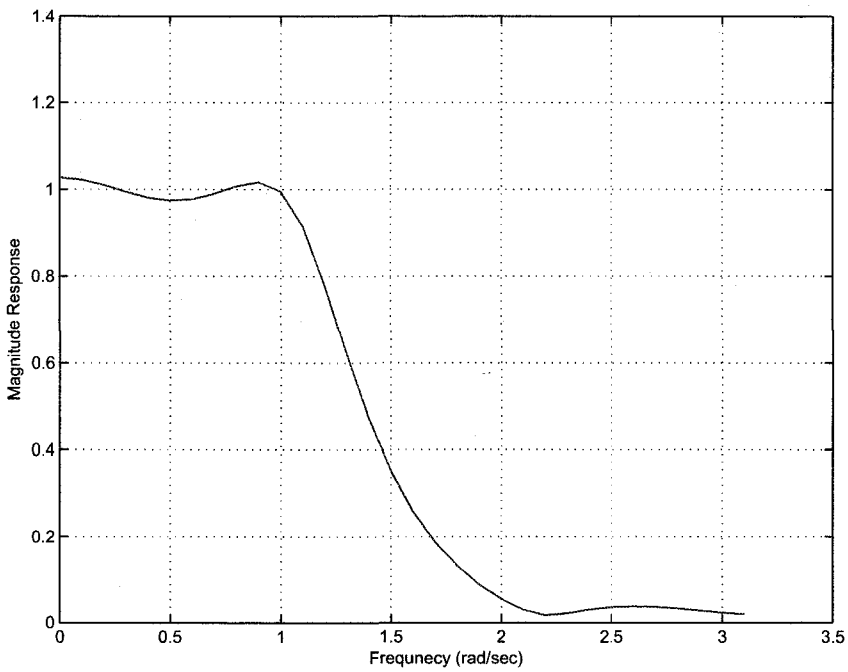


Figure 4.3: 5^{th} order, low-pass IIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Ternary Coding)

Table 4.3: Binary Coefficients of the 5^{th} order low-pass IIR filter with ternary coding

Coefficients	CSD Coefficients
a_0	$2^{-8} + 2^{-15}$
a_1	$2^{-3} - 2^{-6} + 2^{-10} - 2^{-13}$
a_2	$2^{-2} + 2^{-7} - 2^{-10} - 2^{-12}$
a_3	$2^{-2} + 2^{-4} + 2^{-6} + 2^{-15}$
a_4	$2^{-2} + 2^{-5} + 2^{-7}$
a_5	$2^{-3} + 2^{-6} - 2^{-10} - 2^{-11}$
b_0	$2^{-0} - 2^{-7} - 2^{-9} + 2^{-13}$
b_1	$-2^{-3} + 2^{-6} - 2^{-9} + 2^{-11}$
b_2	$2^{-4} + 2^{-8} + 2^{-10} + 2^{-12}$
b_3	$2^{-1} - 2^{-3} - 2^{-7} + 2^{-9}$
b_4	$2^{-5} + 2^{-7} - 2^{-10}$
b_5	$-2^{-3} + 2^{-5} - 2^{-7} - 2^{-13}$

4.2.3.3 FIR filters with new CSD coding scheme

In this example, a 19^{th} order, low-pass FIR filter with the new coding scheme of CSD coefficients and following specification is designed.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.12}$$

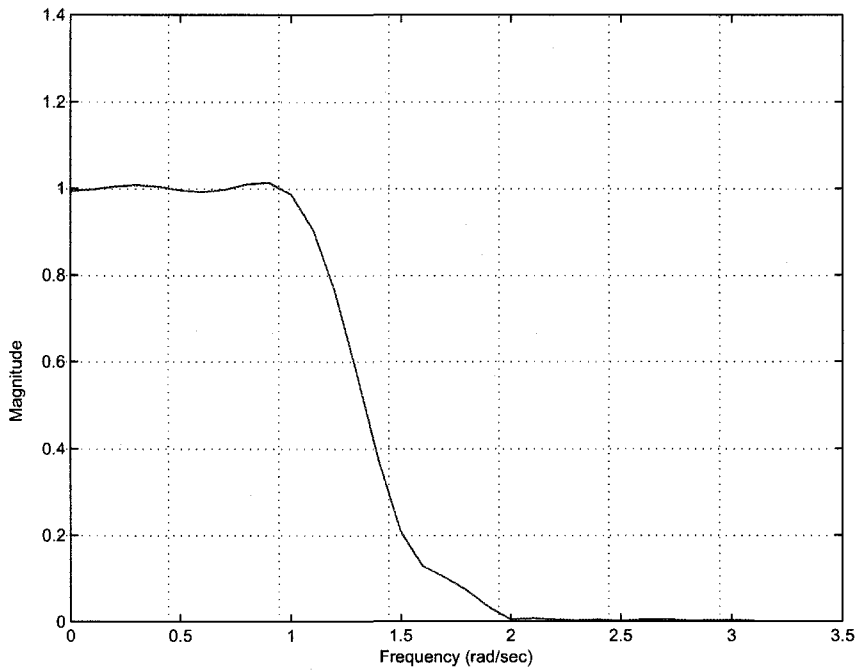


Figure 4.4: 19^{th} order, low-pass FIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (new coding scheme)

Table 4.4: Binary Coefficients of the 19^{th} order low-pass FIR filter with new coding scheme

Coefficients	CSD Coefficients
a_0	$2^{-9} - 2^{-12} + 2^{-15}$
a_1	$2^{-9} + 2^{-11} + 2^{-14}$
a_2	$-2^{-7} - 2^{-9} - 2^{-13}$
a_3	$-2^{-6} + 2^{-8} + 2^{-10} + 2^{-12}$
a_4	$2^{-6} - 2^{-8} + 2^{-10} + 2^{-13}$
a_5	$2^{-5} + 2^{-10} + 2^{-13} + 2^{-15}$
a_6	$-2^{-6} - 2^{-8} - 2^{-10} - 2^{-12}$
a_7	$-2^{-3} + 2^{-5} + 2^{-7} + 2^{-9}$
a_8	$2^{-5} - 2^{-7} - 2^{-13} + 2^{-15}$
a_9	$2^{-2} + 2^{-4} - 2^{-9} - 2^{-12}$
a_{10}	$2^{-1} - 2^{-5} + 2^{-9} - 2^{-11}$
a_{11}	$2^{-2} + 2^{-4} - 2^{-9} - 2^{-12}$
a_{12}	$2^{-5} - 2^{-7} - 2^{-13} + 2^{-15}$
a_{13}	$-2^{-3} + 2^{-5} + 2^{-7} + 2^{-9}$
a_{14}	$-2^{-6} - 2^{-8} + 2^{-10} - 2^{-12}$
a_{15}	$2^{-6} + 2^{-11} + 2^{-13} + 2^{-15}$
a_{16}	$2^{-6} - 2^{-8} + 2^{-10} + 2^{-13}$
a_{17}	$-2^{-6} + 2^{-8} + 2^{-10} - 2^{-12}$
a_{18}	$-2^{-7} + 2^{-9} - 2^{-13}$
a_{19}	$2^{-11} + 2^{-11} + 2^{-14}$

4.2.3.4 IIR filters with new CSD coding scheme

In this example, a 5^{th} order, low-pass IIR filter with the new coding scheme of CSD coefficients and following specification is designed.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.13}$$

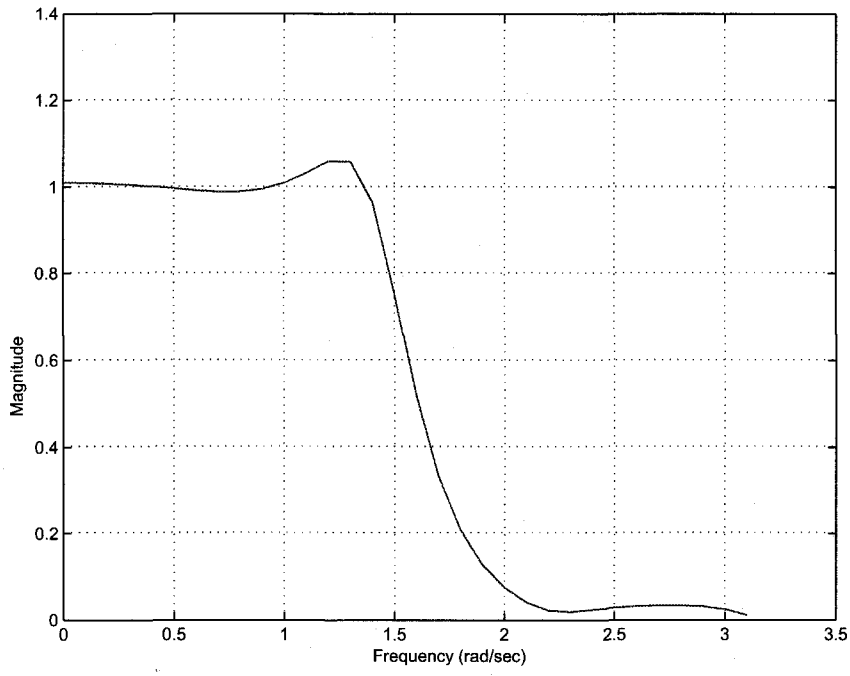


Figure 4.5: 5^{th} order, low-pass IIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (new coding scheme)

Table 4.5: Binary Coefficients of the 5^{th} order low-pass IIR filter with the new coding scheme

Coefficients	CSD Coefficients
a_0	$2^{-4} + 2^{-6} + 2^{-8}$
a_1	$2^{-2} - 2^{-4} + 2^{-7} - 2^{-10}$
a_2	$2^{-1} + 2^{-5} - 2^{-9}$
a_3	$2^{-1} + 2^{-4} + 2^{-7}$
a_4	$2^{-7} + 2^{-12} + 2^{-14}$
a_5	$-2^{-2} + 2^{-5} + 2^{-13}$
b_0	$2^{-0} - 2^{-2} - 2^{-5} + 2^{-8}$
b_1	$2^{-0} - 2^{-7} - 2^{-9} - 2^{-13}$
b_2	$-2^{-3} - 2^{-5} - 2^{-8} - 2^{-10}$
b_3	$2^{-0} + (-2)^{-5} - 2^{-8} - 2^{-10}$
b_4	$2^{-2} - 2^{-12} + 2^{-15}$
b_5	$2^{-5} - 2^{-7} + 2^{-11}$

4.2.3.5 FIR filters with linear phase characteristic and new CSD coding scheme

In this example, a 19^{th} order, low-pass FIR filter with the new coding scheme of CSD coefficients and linear pass-band phase characteristics is designed. following is the specification.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.14}$$

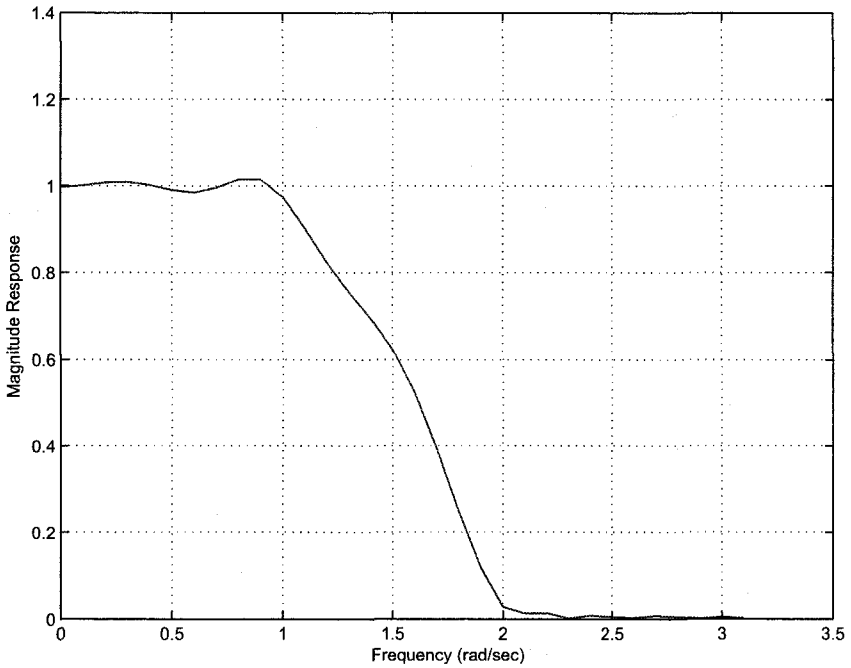


Figure 4.6: 19^{th} order, low-pass FIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Linear Phase and new coding scheme)

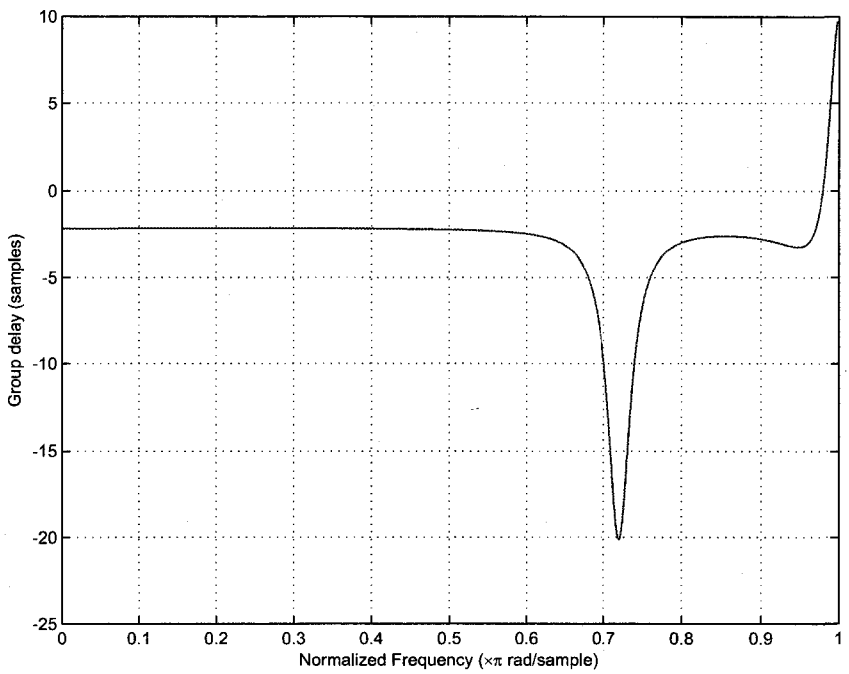


Figure 4.7: Phase Characteristic of 19^{th} order, low-pass FIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Linear Phase and new coding scheme)

4.2.3.6 IIR filters with linear phase characteristic and new CSD coding scheme

In this example, a 5^{th} order, low-pass IIR filter with the new coding scheme of CSD coefficients and linear pass-band phase characteristics is designed. following is the specification.

$$|H(e^{jwT})| = \begin{cases} 1, & 0 \leq w \leq 1.0 \\ 0, & 2.0 \leq w \leq w_s/2 \end{cases} \tag{4.15}$$

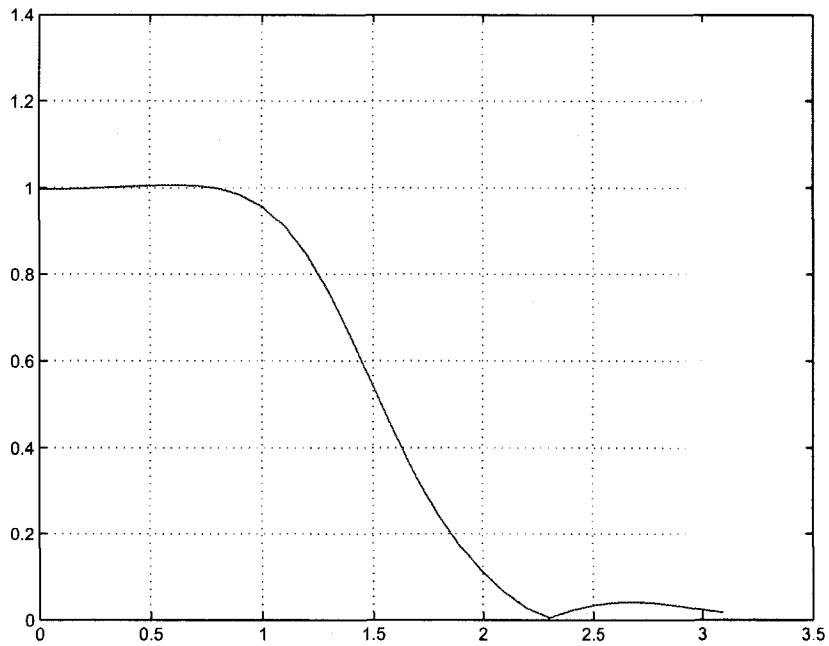


Figure 4.8: 5^{th} order, low-pass IIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Linear Phase and new coding scheme)

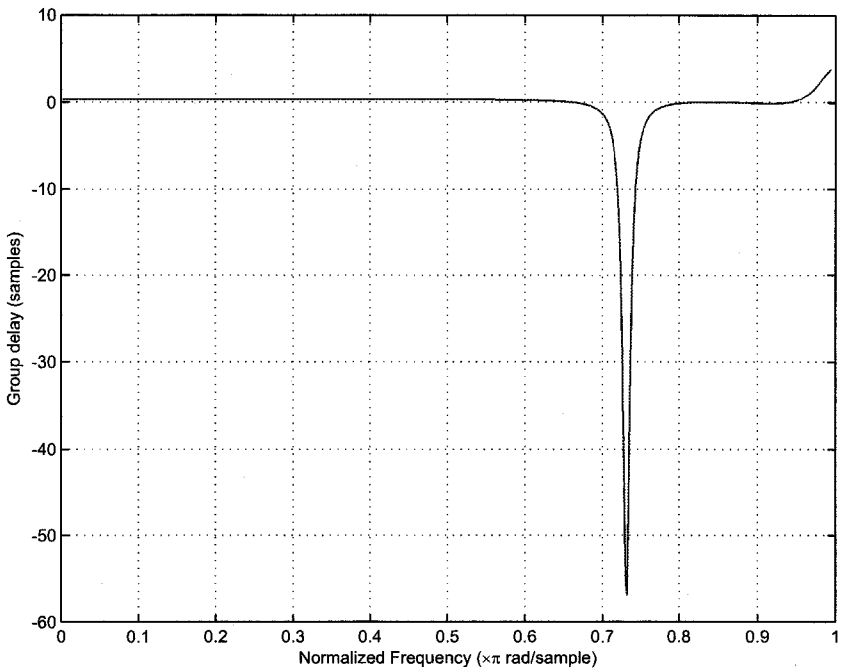


Figure 4.9: Phase Characteristic of 5^{th} order, low-pass IIR filter with 16 bit CSD coefficients and maximum 4 non-zero digits (Linear Phase and new coding scheme)

4.2.3.7 Comparison of Two Coding Schemes

The two coding schemes are used to design IIR and FIR QMF banks. For both coding schemes, the word-length of CSD number is 16.

We start this comparison with IIR QMF bank with the order of 5 for $H_0(z)$. Fig. 4.10(a) shows the filter designed by scheme 1 (ternary coding) and Fig. 4.10(b) shows the filter designed by scheme 2 (new coding scheme), both have the limit of 4 non-zero digits. The experiment is repeated for $N = 3, 4, 5$ maximum non-zero digits and comparison is done with respect to time and mean square error. Simulation results are shown in Table 4.6. According to this table, scheme 2 (new coding scheme) has much better performance than scheme 1 (ternary coding) for IIR QMF filters.

We continued this experiment for FIR QMF banks. Similar to the previous example, experiments were conducted for different number of allowable non-zero digits in CSD format. Fig. 4.11(a) shows the filter designed by scheme 1 (ternary coding) and Fig. 4.11(b) shows the filter designed by scheme 2 (new coding scheme), both have 4 non-zero digits.

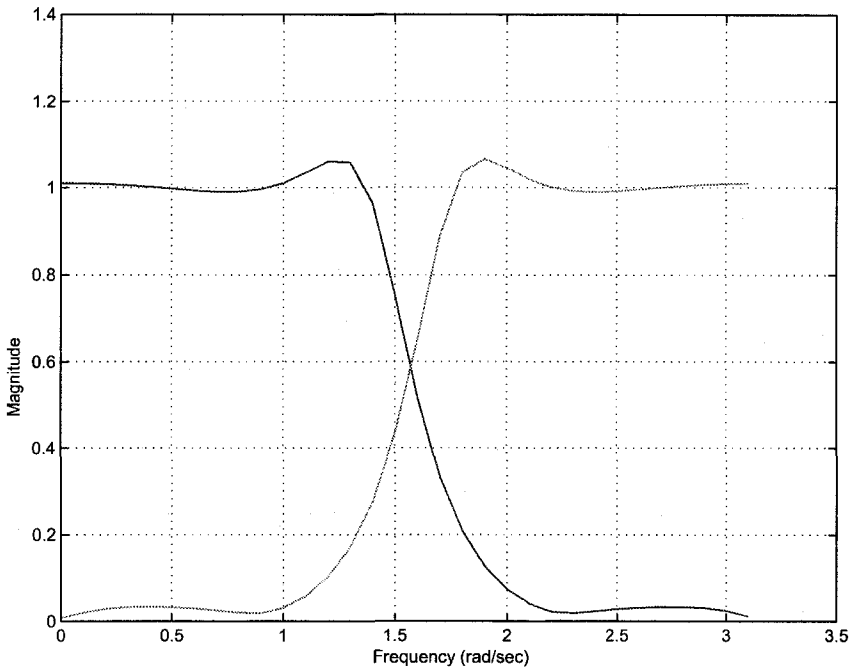
The results are indicated in table 4.7. According to table 4.7, for FIR filters which usually have higher orders, scheme 2 (new coding scheme) has much better performance than scheme 1 (ternary coding).

Table 4.6: Comparison for IIR QMF Bank Order 5

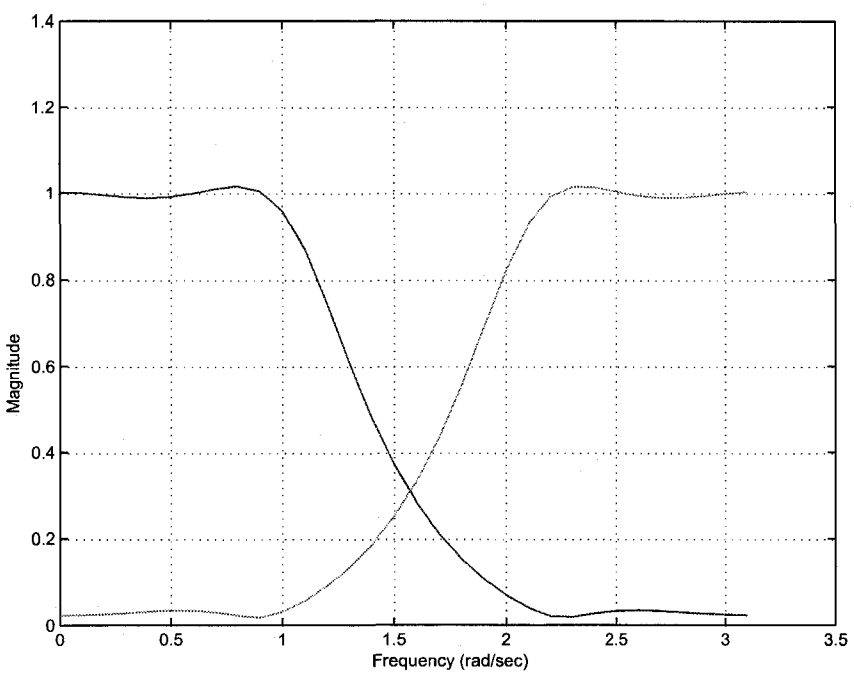
N	Technique 1 (ternary coding)		Technique 2 (new coding scheme)	
	Error	Time (sec)	Error	Time (sec)
3	9.89E-04	121	7.93E-04	117
4	7.20E-04	117	6.66E-04	120
5	2.53E-04	117	1.74E-04	110

Table 4.7: Comparison for FIR QMF Bank Order 19

N	Technique 1 (ternary coding)		Technique 2 (new coding scheme)	
	Error	Time (sec)	Error	Time (sec)
3	7.34E-04	550	1.64E-04	459
4	6.10E-04	564	1.42E-04	447
5	3.00E-04	580	1.06E-04	451

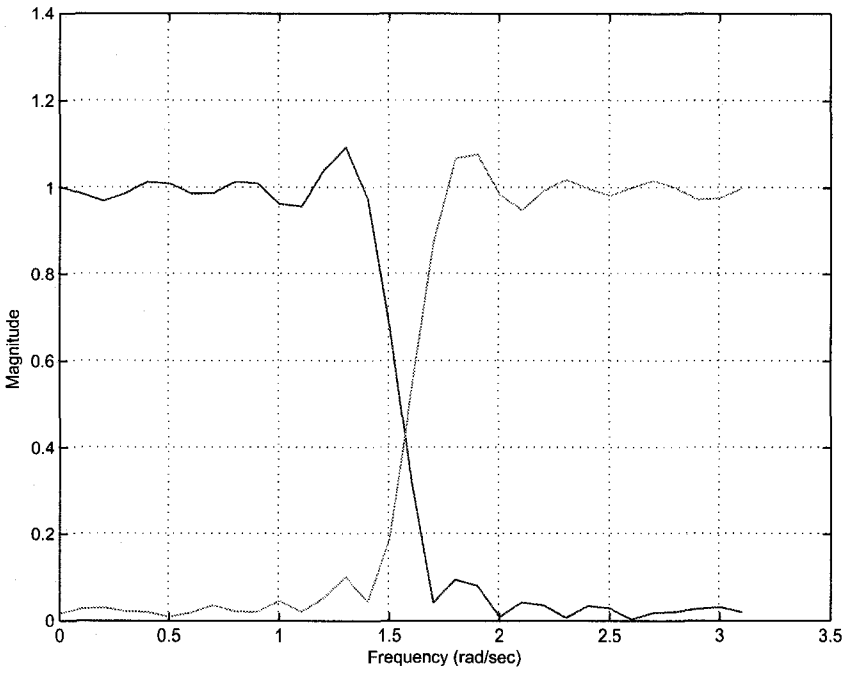


(a) IIR QMF bank designed with ternary coding

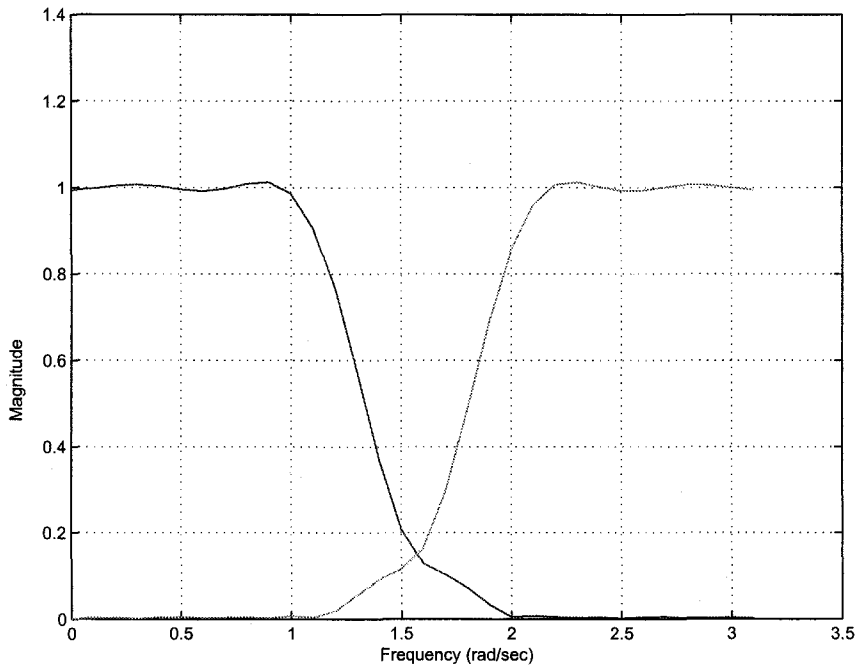


(b) IIR QMF bank designed with new coding scheme

Figure 4.10: Comparison of two techniques for IIR filters



(a) FIR QMF bank designed with ternary coding



(b) FIR QMF bank designed with new coding scheme

Figure 4.11: Comparison of two techniques for FIR filters

4.2.4 Performance Analysis of GA

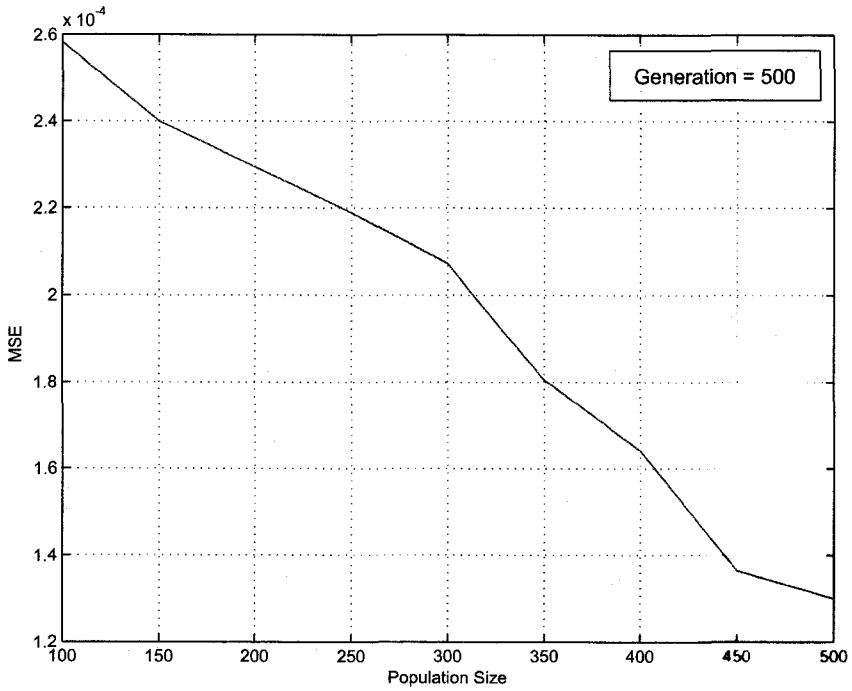
In the previous section, design of following filters were reviewed.

- FIR filters with ternary CSD coding
- IIR filters with ternary CSD coding
- FIR filters with new CSD coding
- IIR filters with new CSD coding
- FIR filters with linear phase characteristic and new CSD coding
- IIR filters with linear phase characteristic and new CSD coding

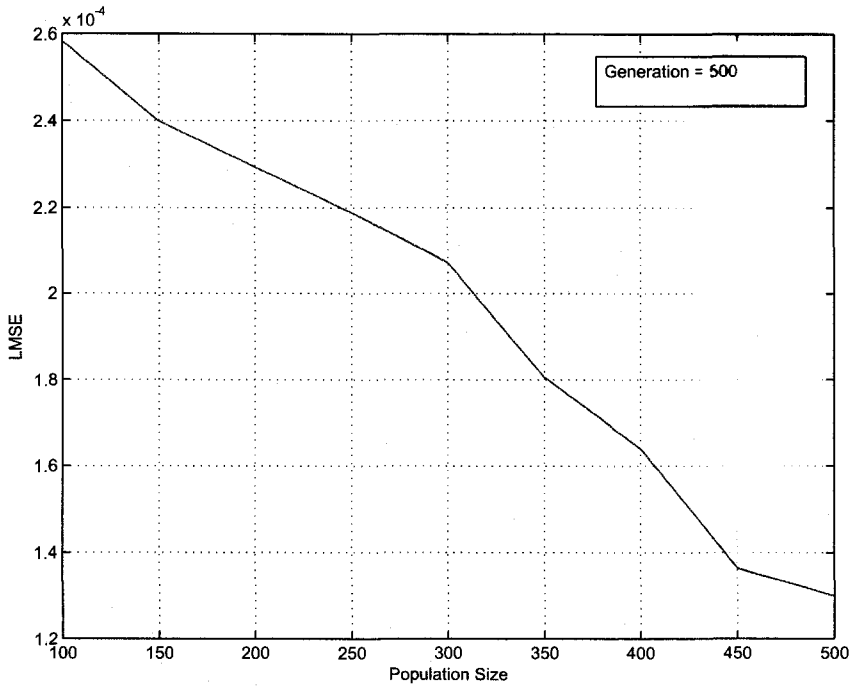
In this section we perform various performance analysis on GA. First the effect of population size and the number of generations on Mean Square Error (MSE) are analyzed. Next, design of digital filters with different crossover techniques is compared and at the end, the effect of P_c and P_m on MSE is examined.

4.2.4.1 Effect of population size and number of generations on MSE

The effect of population size and number of generations on Mean Square Error (MSE) is an important issue in Genetic Algorithm which was overlooked in literature. In this section, two performance analyses of GA on dependence of population size and number of generations to Mean Square Error are presented. In the first one, a minimum value for error is set, and then the number of generations needed to reach the specified error for different sizes of population is measured. As it can be seen from the Fig. 4.12(a), with larger population, less number of generations is needed to reach the desired error. In the second analysis, for defined number of generations and different population size, the MSE is measured. Fig. 4.12(b) shows that, as the population size increases, the Mean Square Error decreases.



(a) No. of Generation vs. Population size for fixed value of error (IIR filter order = 5)



(b) Mean Square Error vs. Population size for fixed number of Generation (IIR filter order = 5)

Figure 4.12: Effect of population size and number of generations on MSE

4.2.4.2 Performance of GA on different crossover techniques

To study the performance of GA with respect to different crossover techniques, digital filters with different orders are designed with these three techniques. Experiments are done on 16-bit 3^{rd}, 5^{th} and 7^{th} order linear phase IIR digital filter with maximum 4 non-zero digits. For considering the random nature of Genetic Algorithm, for each order, 10 filters were designed and the mean error was calculated. As it can be seen from the Table 4.8, 2-point cross-over generally yields better result.

Table 4.8: MSE for 3^{rd}, 5^{th} and 7^{th} order IIR filter with different cross-over techniques

Cross-over	Error		
Techniques	3^{rd} order	5^{th} order	5^{th} order
1-point	1.03E-03	8.21E-04	6.26E-04
2-point	6.95E-04	4.33E-04	3.32E-04
multi-point	1.01E-03	5.76E-04	5.44E-04

4.2.4.3 Effect of P_c and P_m on MSE

Setting proper values for Probability of Crossover (P_c) and Probability of Mutation (P_m) can critically improve the performance of Genetic Algorithm. In this section, two performance analysis of GA on dependence of Probability of Crossover (P_c) and Probability of Mutation (P_m) to Mean Square Error is done. In the first one, with the $P_m = 0.05$, P_c is swept from 65 percent to 100 percent and Mean Square Error is measured. To overcome the random nature of Genetic Algorithm, for each step ten 5^{th} order linear phase IIR digital filter are designed and the mean square error is measured. According to our results, setting the P_c around 95 percent yields to smaller Mean Square Error.

In the second experiment, P_c is fixed at 95 percent and P_m is swept from 0 to 10 percent. In each step, a 5^{th} order linear phase IIR digital filter is designed 10 times

and the MSE is measured. Our experiments show that, having P_m between 4 to 6 percent yields to better results.

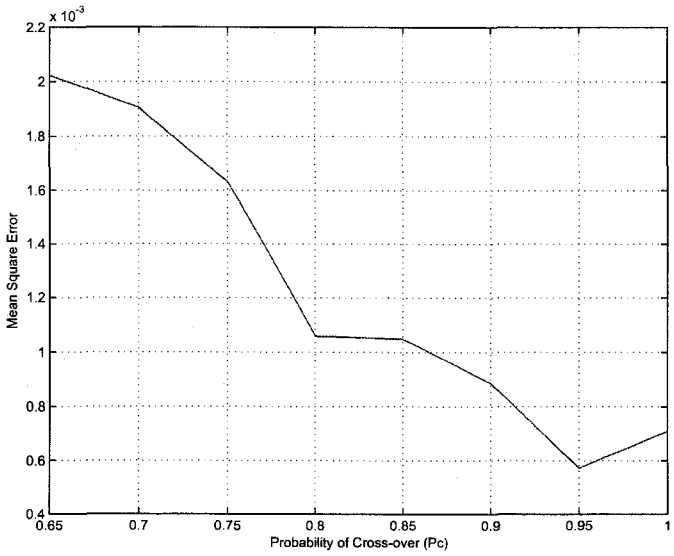


Figure 4.13: Mean Square Error vs. P_c

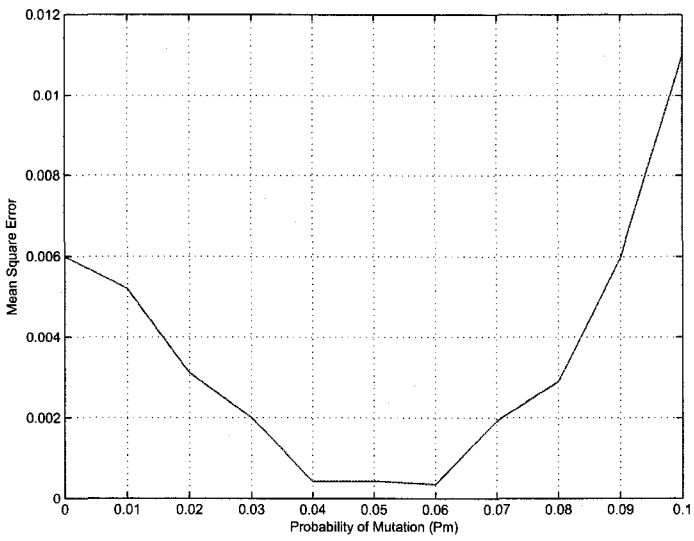


Figure 4.14: Mean Square Error vs. P_m

4.3 Design of Digital Filter with IP

After studying the Immune Programming and its algorithm for solving different problems, we tried to apply IP for the design of digital filters. Fig. 4.15 is the proposed design flow for designing digital filters which is based on IP algorithm. During the experiments the parameters of IP, P_r, P_c, P_m were changed and different types of filters were examined. As it can be seen from Fig. 4.16, a 5^{th} order IIR with 16 bit coefficient and 4 non-zero digits is designed. For the population size of 300 and 500 generations, the results are not comparable to GA.

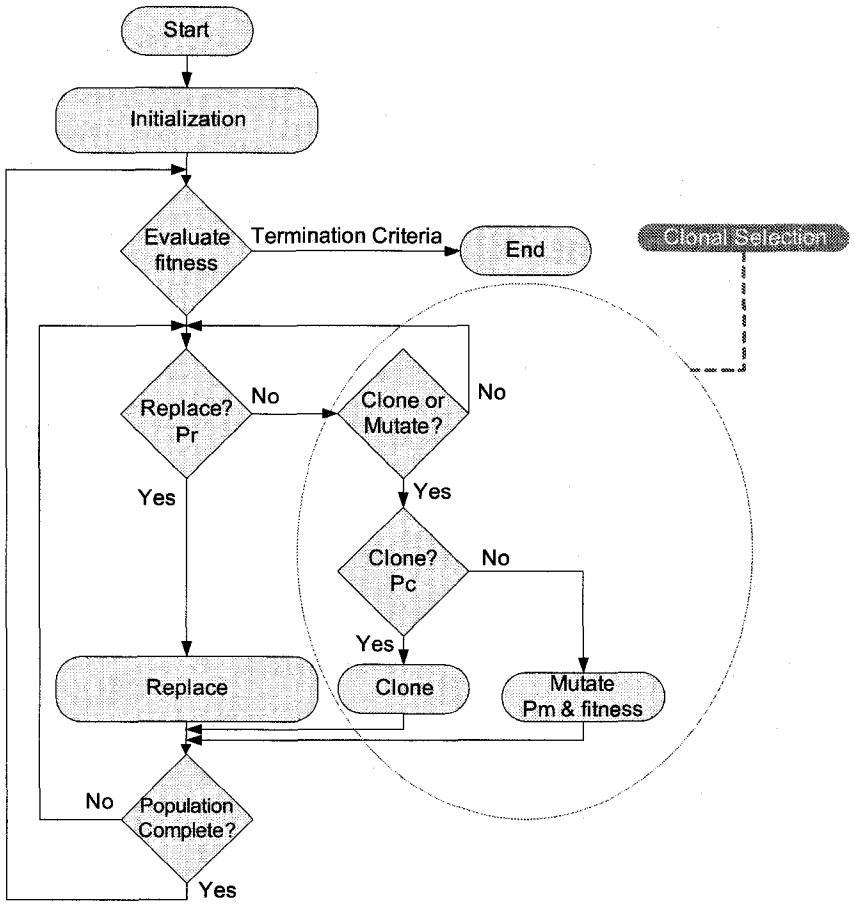


Figure 4.15: IP Design Flow.

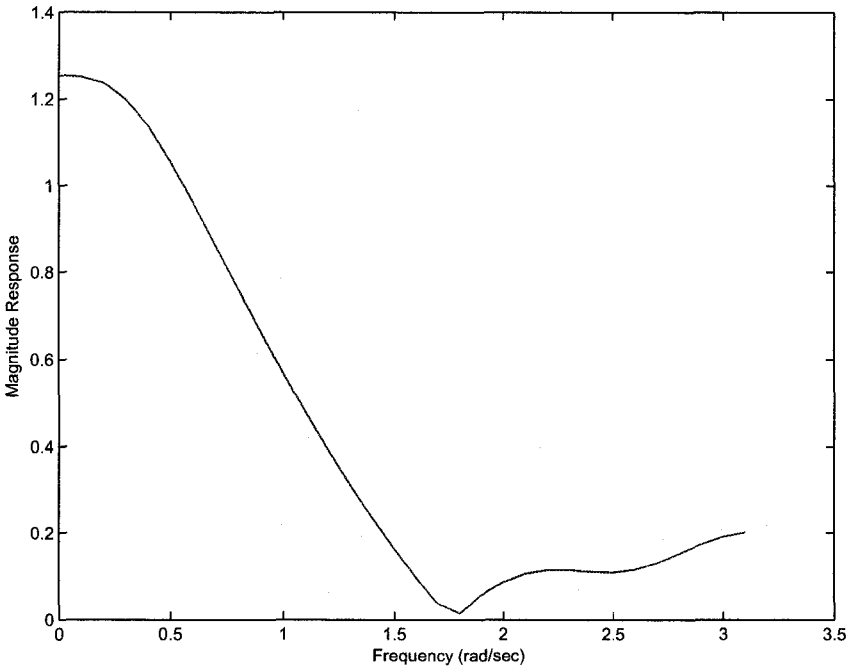


Figure 4.16: 5^{th} order IIR Filter Designed using IP.

We also examined the Immune Programming for finding and eliminating common subexpressions. Fig. 4.17 shows the number of eliminations found by IP for a 19^{th} order FIR filter with 16-bit CSD coefficient and 4 non-zero digits. As it can be seen from the figure, for the repertoire size of 300 and 200 generations, there is no improvement in the number of eliminations.

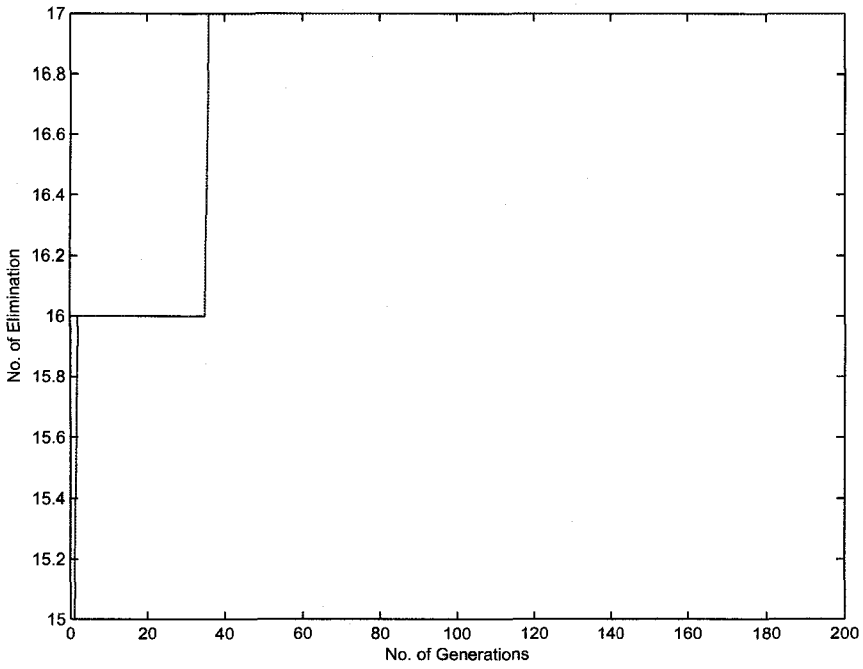


Figure 4.17: Common Subexpression Elimination for a 19^{th} order FIR filter using IP.

4.4 New Algorithm for Digital Filter Design

In this section the proposed algorithm for the design of digital filters with Canonical Signed Digit (CSD) coefficients and Common Subexpression Elimination (CSE) is presented. In this method CSE is added to the fitness function, therefore while searching for the coefficients with the best magnitude and phase characteristics, the number of Common Subexpression Eliminations is also considered.

Fig. 4.18 shows the proposed fitness function. In this algorithm, chromosomes are first examined for the desired magnitude and phase characteristics. For FIR filters there is no need for stability check but for IIR filters, stability check is performed. Next the coefficients undergo Common Subexpression Elimination. The number of reduction in addition for each coefficient is considered as a factor in the fitness function. Equation 4.16 illustrated the proposed error function. α, β and δ are the weighting factors to emphasize the magnitude characteristics, phase characteristics or common subexpression elimination.

$$E = \alpha \sum_{m \in I_{ps}} E_{mag}^2(jw_m) + \beta \sum_{m \in I_p} E_r^2(jw_{m1}) + \delta(CSE) \tag{4.16}$$

$$fitness = \frac{1}{E} \tag{4.17}$$

The new fitness function is utilized for the design of digital filters. In the new design methodology, there are two Genetic Algorithm loops inside each other. The first algorithm tries to find the optimal coefficients for the specified characteristics and the second Genetic Algorithm finds the maximum number of addition elimination in the coefficients. With the new design methodology, in the same time, candidate coefficients are examined for both filter characteristics and CSE reduction, thus more high throughput filters can be achieved.

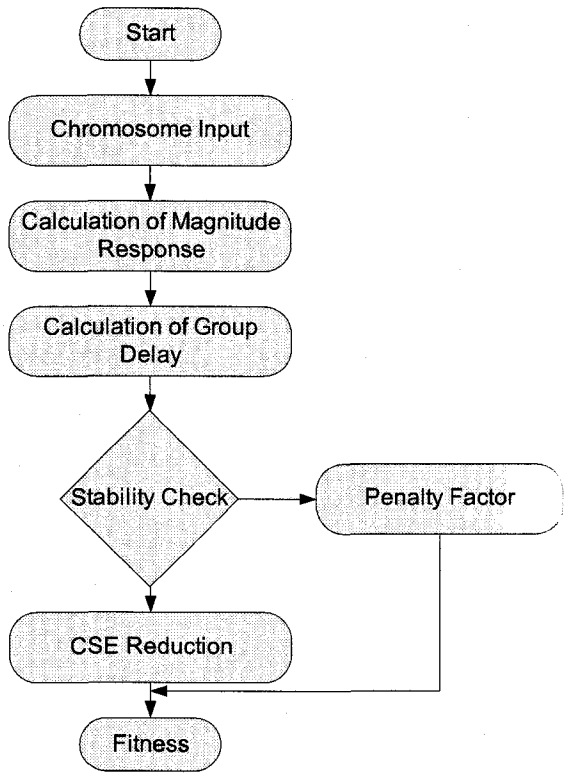


Figure 4.18: The Proposed Fitness Function.

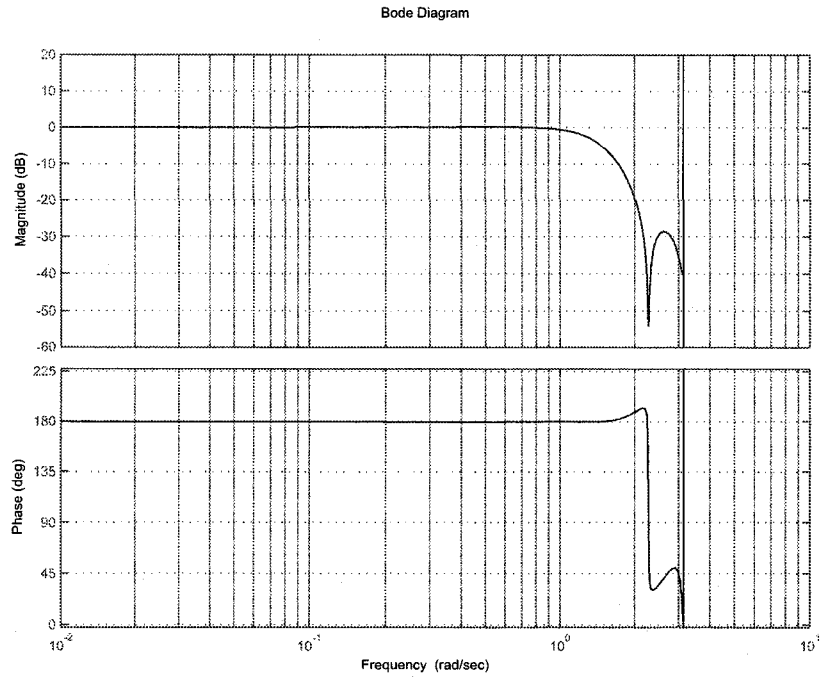
4.4.1 Example

Having introduced the new algorithm for the design of high throughput digital filters, now we design a 5^{th} order low-pass IIR filter (Equation 4.15) with canonical signed digit coefficients, linear pass-band phase characteristics and common subexpression elimination. In this example coefficients are 16 bit and each has a maximum of 4 non-zero digits. Fig. 4.19(a) shows the magnitude and phase response and Fig. 4.19(b) shows the group delay of the designed filter. Originally this filter had 37 additions but with CSE, the number of additions are decreased to 31 which shows 17% efficiency.

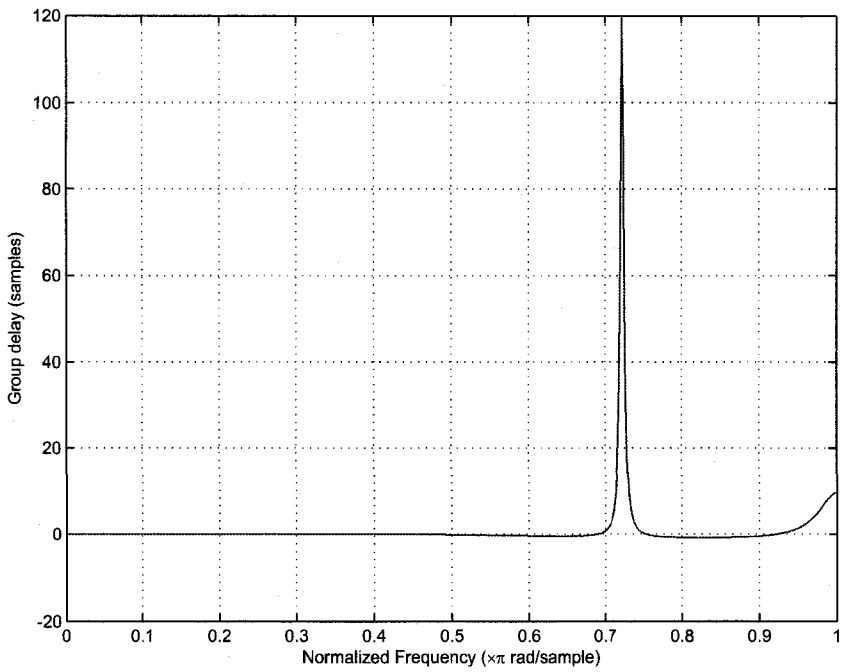
Table 4.9: CSD coefficients of 5^{th} order IIR filter

Canonical Signed Digit Coefficients																
a_0	0	0	0	0	0	-1	0	1	0	0	0	-1	0	0	0	0
a_1	1	0	0	0	0	-1	0	0	-1	0	0	0	0	0	0	0
a_2	0	-1	0	0	0	1	0	0	0	-1	0	0	0	0	0	0
a_3	0	0	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
a_4	0	0	-1	0	0	-1	0	0	0	0	0	0	0	0	0	0
a_5	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	-1	0
b_0	0	0	0	0	0	0	0	1	0	0	0	1	0	-1	0	0
b_1	0	0	0	0	0	1	0	0	-1	0	0	1	0	0	0	-1
b_2	0	0	0	0	0	1	0	0	0	1	0	-1	0	0	0	0
b_3	1	0	0	-1	0	0	0	-1	0	0	-1	0	0	0	0	0
b_4	0	0	0	-1	0	-1	0	0	1	0	0	-1	0	0	0	0
b_5	0	0	0	1	0	0	1	0	0	0	0	0	0	0	-1	0

Coefficients with Subexpression Elimination																
a_0	0	0	0	0	0	+V	0	1	0	0	0	-1	0	0	0	0
a_1	1	0	0	0	0	+V	0	0	-1	0	0	0	0	0	0	0
a_2	0	-1	0	0	0	-3H	0	0	0	-3H	0	0	0	0	0	0
a_3	0	0	+V	0	0	+V	0	0	0	0	0	0	0	0	0	0
a_4	0	0	+V	0	0	+V	0	0	0	0	0	0	0	0	0	0
a_5	0	0	0	0	0	-1	0	-1	0	0	0	0	0	0	-1	0
b_0	0	0	0	0	0	0	0	1	0	0	0	+-V	0	-1	0	0
b_1	0	0	0	0	0	-2H	0	0	-2H	0	0	+-V	0	0	0	-1
b_2	0	0	0	0	0	1	0	0	0	1	0	+-V	0	0	0	0
b_3	1	0	0	+-V	0	0	0	-1	0	0	-1	0	0	0	0	0
b_4	0	0	0	+-V	0	-2H	0	0	1	0	0	-2H	0	0	0	0
b_5	0	0	0	+-V	0	0	1	0	0	0	0	0	0	0	-1	0



(a) Magnitude and Phase Characteristics (IIR filter order = 5)



(b) Group Delay (IIR filter order = 5)

Figure 4.19: 5^{th} order IIR filter designed with the new algorithm

4.5 Conclusion

In this chapter a new algorithm for the design of high throughput digital filters using genetic algorithm was proposed. The new algorithm is a multi objectives optimization technique which examines the magnitude and phase characteristics and also the common subexpression elimination within the canonical signed digit coefficients in its fitness function.

In this chapter, we also presented a graphical method for finding and eliminating 2 non-zero digits common subexpressions in vertical and horizontal positions. We extended this algorithm to 3 non-zero digits in vertical position which led to 31% of reduction in the number of additions.

Chapter 5

Conclusion

In this thesis, we have proposed a new method for the design of high throughput digital filters using Genetic Algorithm.

The proposed design includes FIR and IIR digital filters and also FIR and IIR Quadrature Mirror Filter bank with canonical signed digit coefficients. Furthermore, a new study on performance of Genetic Algorithm with respect to population size, number of generations and fitness function was carried out. Simulation results indicate that for a fixed error, increasing the population size, will reduce the number of generations required.

Also, it was shown that, for a fixed number of generations, by increasing the population size, Mean Square Error is decreased. We also compared two different coding techniques for design of digital filters with CSD coefficient. In technique one, chromosomes are CSD numbers and restoration technique is to convert the violated chromosomes to decimal format and again convert them back to CSD format in the design process. While in the second technique, chromosomes are presented according

to a new format which just indicates the position and sign of non-zero digits. In this technique, non-zero digits which cause violation are simply ignored. According to our experiments for both IIR and FIR filters, technique 2 (new coding scheme) yields better performance.

The next study was in two parts. In the first part, we compared the effect of three different cross-over techniques on Genetic Algorithm. According to our experiments, 2-point cross-over yields to better result for digital filters. In part two, performance of Genetic Algorithm with respect to Probability of Crossover (P_c) and Probability of Mutation (P_m) was analyzed. Simulations show that for linear phase IIR digital filters, having P_c of around 95 percent and P_m of 4 to 6 percent produces the best results.

We also utilized Immune Programming algorithm for the design of digital filters and common subexpression elimination. After trying different filter types with different variations for IP parameters, results obtained were not comparable to Genetic Algorithm.

A new algorithm for Common Subexpression Elimination for reducing the number of addition in Multiple Constant Multiplication for digital filters were developed for 3 non-zero digits in vertical position. In this approach, first common subexpressions in CSD coefficient of digital filters are found and then with Genetic Algorithm the most optimal combination is eliminated. Our simulation results show that by choosing different type of subexpressions (2 or 3 non-zero digit), a 25% to 32% of reduction in number of additions is obtainable.

Finally a new algorithm for the design of high throughput digital filters was proposed. The fitness function of the new algorithm comprises Magnitude Characteristics, Phase Characteristics, Stability check and Common Subexpression Elimination. Through the new fitness function, coefficients are not only examined for the desired filter characteristics but also for the maximum number of common subexpression

elimination, as a result, more high throughput filters can be achieved.

References

[1] A.Lee, M.Ahmadi, G.A.Jullien, R.S.Lashkari, and W.C.Miller. Design of 1-d fir filters with genetic algorithms. In *Proceeding of 1999 IEEE International Symposium on Circuits and Systems*, volume 3, pages 295 - 298. June 1999.

[2] J. Antonisse. A new interpretation of schema notation that overturns the binary encoding constraint. In *Proceeding of third international conference on Genetic Algorithms*, pages 86–91, May 1989.

[3] A.P.Vinod and E.M-K Lai. Comparison of the horizontal and the vertical common subexpression elimination methods for realizing digital filters. In *Proc. of IEEE Int. Symposium on Circuits and Systems (ISCAS)*, volume I, pages 496–499, May 2005.

[4] F. Ashrafzadeh and B. Nowrouzian. Crossover and mutation in genetic algorithms employing canonical signed digit coefficients. In *Proceeding of Midwest symposium on Circuits and Systems*, pages 702–705, Aug. 1997.

[5] A.T.G.Fuller, B.Nowrouzian, and F.Afsharzadeh. Optimization of fir digital filters over canonical signed digit coefficient space using genetic algorithms. In *Proc. of IEEE Midwest Symp. on Circuits and Systems*, pages 456–459, Aug. 1998.

[6] David B.Fogel and Charles J.Robinson. *Computational Intelligence, The expert Speak.* IEEE press, 2003.

[7] D.Dasgupta. *Artificial Immune Systems and their application.* Springer, 1 edition, 1998.

[8] D.R.Forsdyke. The origins of the clonal selection theory of immunity. *FASEB. Journal*, 9, 1995.

[9] Eiben, Agoston E., and Smith. *Introduction to evolutionary computing.* Springer, 2003.

[10] L.J. Eshelman, R. Caruna, and J.D. Schaffer. Biases in the crossover landscape. *In J.D. Schaffer, editor, Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, May 1989.

[11] F.Asharafzadeh, B.Nowrouzian, and A.T.G.Fuller. A novel modified branch-and-bound technique for discrete optimization over canonical signed digit number space. In *Proceeding of IEEE international symposium on circuits and systems*, volume 5, pages 391 – 394, May 1998.

[12] R. Gandhi and S. K. Mitra. A computationally efficient design of two-band qmf banks based on frequency-sampling approach. In *Proceeding of the IEEE international symposium on Circuits and Systems*, volume 3, pages 421–424, June 1998.

[13] G.Gutin and A.P.Punnen. *The Traveling Salesman Problem and Its Variations.* Springer, 2006.

[14] D.E. Goldberg. *Genetic Algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[15] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing with multomodal function optimizations. *In Proceeding of second international conference on genetic algorithm*, pages 41–49, March 1987.

[16] H.D.Tuan, Tran Thai Son, Pierre Apkarian, and Truong Q.Nguyen. Low-order iir filter bank design. In *Proc. of IEEE Transaction on Circuits and Systems*, volume 52, pages 1673 – 1683, Aug. 2005.

[17] H.Safiri, M.Ahmadi, G.A.Jullien, and W.C.Miller. A new algorithm for the elimination of common subexpressions in hardware implementation of digital filters by using genetic programming. In *Proceeding of 2001 IEEE Conference on Application-Specific Systems, Architectures, and Processors*, volume 1, pages 319 – 328, July 2000.

[18] H.Safiri, M.Ahmadi, G.A.Jullien, and W.C.Miller. A novel approach based on genetic algorithm for pipelining of recursive filters. In *Proceeding of 2001 IEEE International Symposium on Circuits and Systems*, volume 2, pages 633 – 636, May 2001.

[19] H.Uppalapati, H.Rastgar, M.Ahmadi, and M.A.Sid-Ahmed. Design of quadrature mirror filter banks with canonical signed digit coefficients using genetic algorithm. In *Proceeding of Fifth International Conference on Communications, Circuits and Systems*, volume 2, pages 27–30, May 2005.

[20] j. H. Holland. *Adaptation in Neural and Artificial Systems.* MIT Press, 1975.

[21] K.Dejong. Genetic algorithms: A 10 year perspective. pages 169–177. Proceedings of the first international conference on genetic algorithms, 1985.

[22] K.S.Tang, K.F.Man, S.Kwong, and Q.He. Genetic algorithms and their applications. *IEEE Signal Processing Magazine*, pages 22–37, Nov. 1997.

[23] Ying-Man Law. Quadrature mirror filter design without transition specification. In *Proc. of IEEE Midwest Symp. on Circuits and Systems*, volume 2, pages 93–96, Jul. 2004.

[24] L.Belfares and A.Guitouni. Multi-objective genetic algorithms for courses of action planning. In *The 2003 congress on evolutionary computing*, volume 3, pages 1543 – 1551, Dec. 2003.

[25] Li Liang. Desing of iir filters with canonical singed digit coefficients using genetic algorithm. Master's thesis, University of Windsor, 2003.

[26] L.Liang, M.Ahmadi, and M.A.Sid-Ahmed. Design of 2d iir filters with canonical signed-digit coefficients using genetic algorithm. In *Proceeding of the 46th IEEE International Midwest Symposium on Circuits and Systems*, volume 2, pages 633 – 635, Dec. 2003.

[27] L.Liang, M.Ahmadi, and M.A.Sid-Ahmed. Design of complementary filter pairs with canonical signed-digit coefficients using genetic algorithm. In *Proceeding of 11th International Conference on Electronics, Circuits and Systems*, volume 1, pages 611 – 614, Dec. 2004.

[28] L.Liang, M.Ahmadi, M.Sid-Ahmed, and K. Wallus. Design of canonical signed digit iir filters using genetic algorithm. In *Proc. of Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, pages 2043 – 2047, Nov. 2003.

[29] P. Mazumder and E. M. Rudnick. *Genetic Algorithms for VLSI Design, Layout and Test Automation.* Prentice Hall, 1998.

[30] M.T.Boraei, M.Ahmadi, S.Erfani, and V.Ramachandran. Design of 2-d recursive digital filters with integer coefficients by successive discritization and re-optimization. In *Proceeding of IEEE Midwest symposium on circuits and systems*, volume 1, pages 294 – 296, May 1985.

[31] Petr Musilek, Adriel Lau, Marek Reformat, and Loren Wyard-Scott. Immune programming. *International Journal of Information Sciences*, 172:972–1001, 2006.

[32] A.Uncini N.Benvenuto, M.Marchesi. Application of simulated annealing for the design of special digital filters. *IEEE Transaction on Signal Processing*, 40:323 – 333, Feb 1992.

[33] Sang Yoon Park and Nam Ik Cho. Design of signed powers-of-two coefficient perfect reconstruction qmf bank using cordic algorithms. *IEEE Transactions On Circuits And Systems-I: Regular Papers*, 53, June 2006.

[34] P.Field. *A Multry Theory for Genetic Algorithms: Unify Binary and Nonbinary Problem Representations.* PhD thesis, University of London, 1996.

[35] R.I.Hartley. Subexpression sharing in filters using canonic signed digit multipliers. *Proc. of IEEE Transactions on Circuits and Systems II*, 43:677–688, Oct. 1996.

[36] R.King, M.Ahmadi, and R.Gorgui-Naguib. *Digital Filtering in One and Two Dimension; Designs and Applications.* Plenum Press, 1989.

[37] R.Pasko, P.Schaumont, V.Derudder, S.Vernalde, and D.Durackova. A new algorithm for elimination of common subexpressions. In *Proc. of IEEE Int. Symposium on Circuits and Systems (ISCAS)*, volume 18, pages 58–68, Jan. 1999.

[38] J. D. Schaffer and L. J. Eshelman. On crossover as an evolutionary viable strategy. In *Proceeding of forth international conference on Genetic Algorithms*, pages 61–68, May 1991.

[39] S.Haykin. *Neural Networks, A Comprehensive Foundation.* IEEE Press, 1 edition, 1994.

[40] S.K.Mitra. *Digital Signal Processing.* McGraw-Hill, 3 edition, 2006.

[41] W.M. Spears. Crossover or mutation? *L. Darrell, editor, Foundations of Genetic Algorithms*, pages 221–237, 1993.

[42] W.M. Spears and K. DeJong. An analysis of multi-point crossover. *G.J.E. Rawlins, editor, Foundations of Genetic Algorithms*, pages 301–315, 1991.

[43] G. Syswerda. Uniform crossover in genetic algorithm. In *Proceeding of third international conference on Genetic Algorithms*, pages 2–9, May 1989.

[44] T.Williams, M.Ahmadi, and W.C.Miller. Genetic algorithms for the design of digital filters using canonic signed digit coefficients. In *Proceeding of 7th International Conference on Signal Processing*, volume 1, pages 9 – 12, Sep. 2004.

[45] V.Ramachandran, M.Ahmadi, and C.S.Gargour. Direct design of recursive digital filters based on a new stability test. *Journal of the Franklin Institute, Pergamon Press Ltd.*, 3:407–413, 1989.

[46] Tom Williams. *Design of High Throughput Recursive and Non-recursive Digital Filters in One and Two Dimensions with Canonical Signed Digit Coefficients and Sub-expression Elimination Using Genetic Algorithm.* PhD thesis, University of Windsor, Sept. 2006.

[47] W.S.Lu and A.Antoniou. *Two Dimensional Digital Filters.* New York, Mercel Dekker, 1992.

[48] Bruce W.Suter. *Multirate and Wavelet Signal Processing.* Academic Press, 1 edition, 1998.

[49] Jinghui Zhong, Xiaomin Hu, Jun Zhang, and Min Gu. Comparison of performance between different selection strategies on simple genetic algorithms. In *Proceeding of international conference on communication, circuits and systems*, volume 2, pages 28–30, Nov. 2005.

[50] Z.Michalewicz. *Genetic Algorithm + Data Structure = Evolution Programs.* Springer, 2 edition, 1992.

[51] Yuanping Zou, Zhengkun Mi, and Minghai Xu. Dynamic load balancing based on roulette wheel selection. In *Proceeding of international conference on communication, circuits and systems*, volume 3, pages 1732 – 1734, June 2006.

VITA AUCTORIS

Payman Samadi was born in Tehran, Iran, in 1982. He received his B.A.Sc. degree in Electrical Engineering in 2005 from Shahid Beheshti University. He is currently Master's candidate in the Electrical and Computer Engineering Department of the University of Windsor. His main research interests include Digital Signal Processing, Communications, VLSI design for Communications and Photonics.