EVALUATION AND ENHANCEMENT OF NATURE-INSPIRED OPTIMIZATION FOR ANTENNA IMPEDANCE MATCHING

A Dissertation in

Electrical Engineering

by

Nathawut Homsup

©2016 Nathawut Homsup

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

December 2016
The dissertation of Nathawut Homsup was reviewed and approved* by the following:

James K. Breakall

Professor of Electrical Engineering

Dissertation Advisor and Chair of Committee

Ram Narayanan

Professor of Electrical Engineering

John J. Metzner

Professor of Electrical Engineering and Computer Science

Kyusun Choi

Associate Professor of Computer Science and Engineering

Kultegen Aydin

Professor and Department Head of Electrical Engineering

*Signatures are on file in the Graduate School
ABSTRACT

Advancements in the technology of communication systems have led to increased requirements and constraints on antennas in terms of cost, bandwidth, and many other performance parameters. In the design of any antenna, a significant amount of time and resources is spent on impedance matching. There are two widely used approaches to impedance matching. The first is to modify the antenna geometry by identifying the appropriate degree of freedom within the structure. The second is the lumped element approach, which modifies the antenna with a passive network consisting of inductors and capacitors to the impedance mismatch between the source and the antenna load. In the second approach, different types of circuits can be used in order to obtain the best results, such as a two lumped elements L network, a T network, a three lumped elements Pi network, or a network with more lumped elements. In order to achieve the desired results from these circuits, we must choose the optimal values of inductor, capacitor, and transducer elements.

To optimize parameters in impedance matching network design, two of the most promising approaches are those of the nature-inspired genetic algorithm (GA) and swarm intelligence algorithms. Examples of these swarm algorithms are particle swarm optimization (PSO), ant colony optimization (ACO), and firefly algorithm (FA). Nature-inspired optimization methods have been used in antenna design for decades in the form of GA and PSO. However, many other algorithms are relatively new to the antenna design problems, even though, these algorithms have already been successfully applied to many other problems and have gained interest in numerous engineering and scientific research fields.

This dissertation compares the described optimization methods by using them to minimize the maximum VSWR over the frequencies 3.5-3.85MHz of the impedance matching network that connects to a high-frequency dipole antenna. In order to satisfy the match over a finite bandwidth, lumped elements were used to construct the matching network, and then adjusting the values of L and C elements.
with the global optimization toolbox from MATLAB, which uses pattern-search, simulated annealing and genetic algorithm, and then compared with the four newer nature-inspired algorithms listed above.

To enhance the nature-inspired algorithms, a new modification of the firefly algorithm (MFA) is proposed. The traditional FA is modified to take into account both the light intensity of and distance between each firefly. This modification results in a more realistic movement of the dimmer firefly towards brighter ones. The proposed algorithm is tested on the traditional FA with eleven standard benchmark functions and on impedance matching network design problems, including both lossless and lossy networks.

The future work for this dissertation is to improve the new MFA by using other techniques such as a hybrid technique with other algorithms. Other more complicated optimization problems can be used to test the MFA.
TABLE OF CONTENTS

LIST OF FIGURES .....................................................................................................................viii

LIST OF TABLES ..........................................................................................................................xiv

ACKNOWLEDGEMENTS ................................................................................................................xv

CHAPTER 1 INTRODUCTION TO OPTIMIZATION ALGORITHMS .................................................1

1.1 Optimization Algorithms ......................................................................................................2
1.2 Brief History of Nature-Inspired Optimization Algorithms ................................................19
1.3 Motivation .............................................................................................................................20
1.4 Organization of the Dissertation ..........................................................................................21

CHAPTER 2 NATURE-INSPIRED OPTIMIZATION METHODS ......................................................22

2.1 Introduction of Nature-Inspired Optimization Methods ........................................................23
2.2 Genetic Algorithm (GA) ......................................................................................................24
    2.2.1 History of Genetic Algorithm (GA) .............................................................................24
    2.2.2 Search Spaces and Elements .....................................................................................26
    2.2.3 Simple Genetic Algorithm (SGA) .............................................................................28
2.3 Particle Swarm Optimization (PSO) ....................................................................................31
    2.3.1 History of Particle Swarm Optimization (PSO) .........................................................31
    2.3.2 Classical PSO .............................................................................................................31
    2.3.3 Selection of Parameters for PSO ..............................................................................34
    2.3.4 Neighborhood Topologies in PSO ..........................................................................36
2.4 Any Colony Optimization (ACO) .......................................................................................38
    2.4.1 Ant System (AS) .......................................................................................................38
    2.4.2 MAX-MIN Ant System (MMAS) ..............................................................................39
    2.4.3 Ant Colony System (ACS) .......................................................................................40
    2.4.4 Standard Ant Colony Optimization ..........................................................................41
    2.4.5 Ant Colony Optimization for Continuous Variable Domains ..................................43
    2.4.6 Modified Continuous Ant Colony Optimization using Prospect Theory ............45
2.5 Firefly Algorithm (FA) .......................................................................................................48
    2.5.1 Introduction to Firefly Algorithm ..............................................................................48
5.5 Test Results and Analysis ................................................................. 119

CHAPTER 6 MODIFIED FIREFLY ALGORITHM (MFA) OPTIMIZATION IN IMPEDANCE MATCHING NETWORKS ........................................................................... 122

6.1 MFA Optimization for Fano LCR Load .................................................. 123
6.2 MFA Optimization for Lossless Impedance Matching network ...................... 127
6.3 MFA Optimization for Lossy Impedance Matching Network ......................... 129
6.4 MFA Optimization for HF Broadband Fan Antenna ..................................... 145

CHAPTER 7 CONCLUSION ............................................................................ 151

7.1 Dissertation Contributions ...................................................................... 151
7.2 Future Research Directions ..................................................................... 153

BIBLIOGRAPHY ......................................................................................... 154

APPENDIX A DIPOLE ANTENNA IMPEDANCE FILE ........................................ 159

APPENDIX B FAN ANTENNA IMPEDANCE FILE ............................................ 160

APPENDIX C MATLAB-BASED CODE FOR GA ........................................... 161

APPENDIX D MATLAB-BASED CODE FOR PSO ........................................ 165

APPENDIX E MATLAB-BASED CODE FOR ACO ......................................... 169

APPENDIX F MATLAB-BASED CODE FOR FA ........................................... 176

APPENDIX G MATLAB-BASED CODE FOR MFA ......................................... 180

APPENDIX H MATLAB-BASED CODE FOR TESTING FUNCTIONS .................. 188
LIST OF FIGURES

Figure 1-1: Feasible regions of LP and ILP.................................................................13

Figure 1-2: Optimization over a high-dimensional cone of a pair of consecutive Boltzman
distribution $e^{-cx/T}$ .................................................................................................16

Figure 1-3: Optimization over a high-dimensional cone of a pair of uniform distribution over
truncated cones $c.x \leq T$. .........................................................................................16

Figure 1-4: A simple neural network equivalent to a linear regression .........................17

Figure 1-5: A neural network with four inputs and one hidden layer with three hidden neurons. 18

Figure 2-1: Pseudo-code of Genetic Algorithm (GA)..................................................30

Figure 2-2: Pseudo-code of Particle Awarm Algorithm (PSO).......................................33

Figure 2-3: Topologies of Particle Swarm Optimization (PSO).....................................37

Figure 2-4: Basic principles of Ant Colony Optimization (ACO).................................42

Figure 2-5: Discrete probability distribution and continuous probability function of an
optimization function .................................................................................................44

Figure 2-6: The structure of the solution achieves. The solutions in the archive are sorted
according to their quality. .........................................................................................45

Figure 2-7: Pseudo-code of ACOR-PT algorithm. .......................................................47

Figure 2-8: Pseudo-code of the Firefly Algorithm (FA)................................................51

Figure 3-1: A load connected to a transmission line through a matching network............55

Figure 3-2: Two-ports broadband matching network...................................................56

Figure 3-3: The matching network is inserted between the source and load...................58

Figure 3-4: The reflection coefficient in ideal situation................................................60

Figure 3-5: Impedance matching networks with reactive loads.....................................61

Figure 3-6: L network topology ..................................................................................63

Figure 3-7: T network topology ..................................................................................64

Figure 3-8: Pi network topology ..................................................................................64

Figure 3-9: Typical circuit configuration with a lossy two-ports network between the source and
the load ......................................................................................................................65
Figure 4-1: Pi3 network .................................................................69

Figure 4-2: VSWR over 3.4-4 MHz with 100 loops using pattern search from the MATLAB global optimization toolbox .................................................................72

Figure 4-3: VSWR over 3.4-4 MHz with 5000 loops using pattern search from the MATLAB global optimization toolbox .................................................................72

Figure 4-4: VSWR over 3.4-4 MHz with 100 loops using simulate annealing from the MATLAB global optimization toolbox .................................................................73

Figure 4-5: VSWR over 3.4-4 MHz with 5000 loops using simulate annealing from the MATLAB global optimization toolbox .................................................................73

Figure 4-6: VSWR over 3.4-4 MHz with 100 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20. .................................................................75

Figure 4-7: VSWR over 3.4-4 MHz with 500 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20. .................................................................75

Figure 4-8: VSWR over 3.4-4 MHz with 1000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20 ..................................................................76

Figure 4-9: VSWR over 3.4-4 MHz with 5000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20 ..................................................................76

Figure 4-10: VSWR over 3.4-4 MHz with 100 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200 .................................................................77

Figure 4-11: VSWR over 3.4-4 MHz with 500 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200 .................................................................78

Figure 4-12: VSWR over 3.4-4 MHz with 1000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200 .................................................................78

Figure 4-13: VSWR over 3.4-4 MHz with 5000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200 .................................................................79

Figure 4-14: VSWR over 3.4-4 MHz using GA with a parameter population = 70, generation = 150 .........................................................................................................................81

Figure 4-15: VSWR over 3.4-4 MHz using GA with a parameter population = 150, generation = 150 .........................................................................................................................81

Figure 4-16: VSWR over 3.4-4 MHz using PSO with a parameter population = 70, iteration = 150 .........................................................................................................................83

Figure 4-17: VSWR over 3.4-4 MHz using PSO with a parameter population = 150, iteration = 150 .........................................................................................................................83

Figure 4-18: VSWR over 3.4-4 MHz using ACO with a parameter population = 70, iteration = 150 .........................................................................................................................85
Figure 4-19: VSWR over 3.4-4 MHz using ACO with a parameter population = 150, iteration = 150.................................................................85

Figure 4-20: VSWR over 3.4-4 MHz using FA with a parameter population = 70, iteration = 150.................................................................87

Figure 4-21: VSWR over 3.4-4 MHz using FA with a parameter population = 150, iteration = 150.................................................................87

Figure 5-1: The Ackley function..................................................................................................................99

Figure 5-2: Locations of fireflies after 10 iterations. ....................................................................................99

Figure 5-3: Final locations of fireflies after 50 iterations, converging into (0,0) ......................100

Figure 5-4: Comparison of the performance of SFA and MFA for minimization of the Ackley function with $\gamma_0 = 25$, and $\alpha_0 = 0.02$. .................................................................100

Figure 5-5: The sphere function..................................................................................................................101

Figure 5-6: Locations of fireflies after 10 iterations. ....................................................................................102

Figure 5-7: Final locations of fireflies after 50 iterations, converging into (0,0) ......................102

Figure 5-8: Comparison of the performance of SFA and MFA with $\gamma_0 = 150$, and $\alpha_0 = 0.3$. .................................................................103

Figure 5-9: The Rastrigin function..............................................................................................................103

Figure 5-10: Locations of fireflies after 10 iterations. ....................................................................................104

Figure 5-11: Final locations of fireflies after 50 iterations, converging into (0,0) ......................104

Figure 5-12: Comparison of the performance of SFA and MFA with $\gamma_0 = 30$, $\alpha_0 = 0.1$, and $\text{Max}_\text{it} = 500$.........................................................................................105

Figure 5-13: The Michalewicz function....................................................................................................106

Figure 5-14: Locations of fireflies after 10 iterations ..................................................................................106

Figure 5-15: Final locations of fireflies after 50 iterations, converging into (2.20,1.57)............107

Figure 5-16: Comparison of the performance of SFA and MFA with $\gamma_0 = 20$, and $\alpha_0 = 0.02$. 107

Figure 5-17: The Yang’s forest function..................................................................................................108

Figure 5-18: Locations of fireflies after 10 iterations ..................................................................................109

Figure 5-19: Final locations of fireflies after 50 iterations, converging into (0,0) ......................109

Figure 5-20: Comparison of the performance of SFA and MFA with $\gamma_0 = 10$, $\alpha_0 = 0.02$. 110
Figure 5-21: The four-peaks function

Figure 5-22: Locations of fireflies after 10 iterations

Figure 5-23: Final locations of fireflies after 50 iterations, converging into (-0.5, -0.5), (0.5, -0.5), (-0.5, 0.5), and (0.5, 0.5)

Figure 5-24: Comparison of performance of the SFA and MFA for maximization of the four-peaks function with $\gamma_0 = 10$ and $\alpha_0 = 0.02$

Figure 5-25: Comparison of performance of the SFA and MFA for minimization of Styblinski-Tang function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 55$, $\text{Max_it} = 70$

Figure 5-26: Comparison of performance of the SFA and MFA for minimization of Helical-Valley function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 55$, $\text{Max_it} = 70$

Figure 5-27: Comparison of performance of the SFA and MFA for minimization of Wood function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 60$, $\text{Max_it} = 90$

Figure 5-28: Comparison of performance of the SFA and MFA for minimization of Powell function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 60$, $\text{Max_it} = 90$

Figure 5-29: Comparison of performance of the SFA and MFA for minimization of Rosenbrock function with $\gamma_0 = 100$, $\alpha_0 = 0.1$, $N = 500$, $\text{Max_it} = 20$

Figure 6-1: Fano LCR load and equalizer topology

Figure 6-2: Optimized equalizer TPG with CFO and MFA

Figure 6-3: Optimized TPG results by CFO with a standard objective function and MFA with a new objective function

Figure 6-4: Lossless Pi3 network

Figure 6-5: VSWR of the optimized Pi3 network, $\text{Min_vswr} = 1.1107$, and $\text{Max_vswr} = 1.3386$

Figure 6-6: Minimum of Max_VSWR vs iteration, comparison results between FA and MFA

Figure 6-7: Lossy network topology with one resistor

Figure 6-8: Lossy network topology with two resistors

Figure 6-9: Lossy network topology with three resistors

Figure 6-10: Lossy Pi3 network

Figure 6-11: VSWR of lossy Pi3 network with $\text{weight}(\lambda) = 0$

Figure 6-12: TPG of lossy Pi3 network with $\text{weight}(\lambda) = 0$

Figure 6-13: VSWR of lossy Pi3 network with $\text{weight}(\lambda) = 0.1$
Figure 6.14: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.1.................................133
Figure 6.15: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.2.................................133
Figure 6.16: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.2.................................134
Figure 6.17: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.3.................................134
Figure 6.18: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.3.................................135
Figure 6.19: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.4.................................135
Figure 6.20: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.4.................................136
Figure 6.21: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.5.................................136
Figure 6.22: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.5.................................137
Figure 6.23: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.6.................................137
Figure 6.24: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.6.................................138
Figure 6.25: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.7.................................138
Figure 6.26: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.7.................................139
Figure 6.27: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.8.................................139
Figure 6.28: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.8.................................140
Figure 6.29: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.9.................................140
Figure 6.30: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.9.................................141
Figure 6.31: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 1.................................141
Figure 6.32: TPG of lossy Pi3 network with weight(\(\lambda\)) = 1.................................142
Figure 6.33: TPG of lossy Pi3 network for \(\lambda=0\), \(\lambda=0.2\), \(\lambda=0.8\), and \(\lambda=1\).................................142
Figure 6.34: VSWR of lossy Pi3 network for \(\lambda=0\), \(\lambda=0.2\), \(\lambda=0.8\), and \(\lambda=1\).................................143
Figure 6.35: CAD frigate model with the HF fan antenna.................................145
Figure 6.36: Radiation impedance of the fan antenna.................................146
Figure 6.37: Matching network topology for this example.................................146
Figure 6.38: VSWR of HF Broadband Fan Antenna for \(\lambda=0\), \(\lambda=0.2\), \(\lambda=0.8\), and \(\lambda=1\).................................149
Figure 6.39: TPG of HF Broadband Fan Antenna for \(\lambda=0\), \(\lambda=0.2\), \(\lambda=0.8\), and \(\lambda=1\).................................149
Figure 6-40: VSWR of HF Broadband Fan Antenna before and after matching...................150
LIST OF TABLES

Figure 4-1: VSWR, time, and lumped elements value result of loop optimizer by pattern search and simulated annealing from the MATLAB global optimization toolbox ..........................71

Figure 4-2: VSWR, time, and lumped elements value result of Loop optimizer by GA with population = 20 from the MATLAB global optimization toolbox.................................74

Figure 4-3: VSWR, time, and lumped elements value results of a loop optimizer by GA with population = 200 from the MATLAB global optimization toolbox..............................77

Figure 4-4: VSWR, time, and lumped elements value result by GA .....................................80

Figure 4-5: VSWR, time, and lumped elements value results by PSO......................................82

Figure 4-6: VSWR, time, and lumped elements value results by ACO.....................................84

Figure 4-7: VSWR, time and lumped elements value results by FA .......................................86

Figure 4-8: VSWR and accuracy results from simulation of GA, PSO, ACO, and FA, run 1000 times each.........................................................................................................................88

Figure 6-1: Optimal component value results of lossy Pi3 network for 11 weight values (\(\lambda = 0, 0.1, \ldots, 1\)). \(\gamma_0 = 25, \alpha_0 = 0.05\), population = 300, and Max_iteration = 20...........131

Figure 6-2: Minimum VSWR, maximum VSWR, minimum TPG, and maximum TPG results of lossy Pi3 network for 11 weight values (\(\lambda = 0, 0.1, \ldots, 1\)) .................................................................143

Figure 6-3: Optimal component value results of matching network of HF Broadband Fan Antenna for 11 weight values (\(\lambda = 0, 0.1, \ldots, 1\)). \(\gamma_0 = 100, \alpha_0 = 0.6\), population = 200, and Max_iteration = 50.................................................................148

Figure 6-4: Minimum VSWR, maximum VSWR, minimum TPG, and maximum TPG results of HF Broadband Fan Antenna for 11 weight values (\(\lambda = 0, 0.1, \ldots, 1\)). with \(\gamma_0 = 100, \alpha_0 = 0.6\), population = 200, and Max_iteration = 50.................................................................148
ACKNOWLEDGEMENTS

I would like to thank all those who have assisted me in completing this work.

Foremost, I would like to express my sincere gratitude to my advisor, Professor James K. Breakall, for his continuous support and guidance through all my Ph.D. study and research. His support for my transition from another research area has been greatly important. He encouraged me to explore various topics in the antenna design research field and gave me many opportunities including experience in his lab, his courses, and antenna field work.

I would like to thank Prof. Ram Narayanan, Prof. John J. Metzner, and Assoc.Prof. Kyusun Choi for being my Ph.D. dissertation committee members. Thank you for all your valuable feedback, comments and suggestions. I appreciate your time evaluating my dissertation, which helped me to improve my work.

I want to thank my parents for the support they have provided me with through my entire life. For being the best example of how to live my life, I always try my best to not disappoint either of you.

Thank you to the Pennsylvania State University for the opportunity for allowing me to study in the graduate program and to faculty members who taught me valuable courses that helped me to expand my knowledge. Thank you also to all of the staff members in the Electrical Engineering department who helped me through all my technical problems.
Chapter 1

Introduction to Optimization Algorithms

In mathematics, the optimization problem describes the problem of finding the best solution from all feasible solutions. Optimization problems can be divided into two categories depending on whether the variables are continuous or discrete. Optimization problems with discrete variables are known as combinational optimization problems. Problems with continuous variables include constrained problems and multimodal problems. Unless, both the objective function and feasible region are convex in a minimization problem, there may be several local minima.

The standard form of continuous optimization is

$$\min_x f(x) \text{ subject to } g_i(x), i = 1, \ldots, m$$

$$h_i(x), i = 1, \ldots, p$$

(1-1)

where \( f(x): R^n \to R \) is the objective function to be minimized over the variable \( x \), \( g_i(x) \leq 0 \) are called inequality constraints, and \( h_i(x) = 0 \) are called equality constraints.

Many of the algorithms proposed for solving non-convex problems including the majority of commercially available solvers, are not capable of making a distinction between local optimal solutions and global optimal solutions, and sometime treat the former as the actual solution to the problem. The type of numerical analysis that is used in the development of deterministic algorithms that are capable of guaranteeing convergence in a finite time to the global optimal solution of a non-convex problem is called global optimization.
1.1 Optimization Algorithms

The existence of optimization methods can be traced to the days of Newton, Lagrange, and Cauchy [1]. The development of different calculus methods for optimization was possible because of the contributions Newton and Leibnitz made to calculus. The foundations of the calculus of variations, which deals with the minimization of functions, were laid by Bernoulli Euler, Lagrange, and Weistrass [2]. The method of optimization for constrained problems, which involves the addition of unknown multipliers, became known by the name of its inventor as the method of *Lagrange multipliers*.

**Lagrange Multipliers**

When we want to maximize or minimize a multivariable function $f(x, y, \ldots)$ subject to the constraint that another multivariable function equals a constant, $g(x, y, \ldots) = c$, that follow these steps:

1) Introduce a new variable $\lambda$ and define a new function $L$ as follows:

$$L(x, y, \ldots, \lambda) = f(x, y, \ldots) - \lambda(g(x, y, \ldots) - c)$$  \hspace{1cm} (1-2)

This function $L$ is called the "Lagrangian", and the new variable $\lambda$ is referred to as a "Lagrange multiplier".

2) Set the gradient of $L$ equal to the zero vector.

$$\nabla L(x, y, \ldots, \lambda) = 0$$  \hspace{1cm} (1-3)

In other words, find the critical points of $L$.

3) Consider each solution, which will look something like $(x_0, y_0, \ldots, \lambda_0)$. Substitute each one into $f$, or rather, first remove the $\lambda_0$ component, then plug it into $f$, since $f$ does not have $\lambda$ as
an input. Whichever solutions gives the greatest (or smallest) value is the maximum (or minimum) point being sought.

**Steepest Descent Method**

In 1847, Cauchy [3] made the first application of the *steepest descent* method to solve unconstrained optimization problems. Later, it was Debye [4] who used this method to estimate Bessel functions. The steepest descent method can be described briefly as below.

Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that is differentiable at $x_0$, the direction of steepest descent is the vector $-\nabla f(x_0)$. To see this, consider the function

$$\varphi(t) = f(x_0 + tu)$$

where $u$ is a unit vector; that is, $||u|| = 1$. Then, by the chain rule,

$$\varphi'(t) = \frac{\partial f}{\partial x_1} \frac{\partial x_1}{\partial t} + \cdots + \frac{\partial f}{\partial x_n} \frac{\partial x_n}{\partial t}$$

$$\varphi'(t) = \frac{\partial f}{\partial x_1} u_1 + \cdots + \frac{\partial f}{\partial x_n} u_n$$

$$\varphi'(t) = \nabla f(x_0 + tu) \cdot u$$

and therefore,

$$\varphi'(0) = \nabla f(x_0) \cdot u = ||\nabla f(x_0)|| \cos \theta$$

where $\theta$ is the angle between $\nabla f(x_0)$ and $u$. It follows that $\varphi'(0)$ is minimized when $\theta = \pi$, which yields

$$u = -\frac{\nabla f(x_0)}{\nabla f(x_0)}$$
\[ \varphi'(0) = -\|\nabla f(x_0)\| \]  

(1-10)

We can therefore reduce the problem of minimizing a function of several variables to a single variable minimization problem, by finding the minimum of \( \varphi(t) \) for this choice of \( u \). That is, we find the value of \( t \), for \( t > 0 \), that minimizes

\[ \varphi_0(t) = f(x_0 - t\nabla f(x_0)) \]  

(1-11)

After finding the minimizer \( t_0 \), we set

\[ x_1 = x_0 - t_0\nabla f(x_0) \]  

(1-12)

Continue the process, by searching from \( x_1 \) in the direction of \( -\nabla f(x_1) \) to obtain \( x_2 \) by minimizing \( \varphi_1(t) = f(x_1 - t\nabla f(x_1)) \), and so on.

Given an initial guess \( x_0 \), the method of steepest descent computes a sequence of iterates \( \{x_k\} \), where

\[ x_{k+1} = x_k - t_k\nabla f(x_k) \]  

(1-13)

\( t_k > 0 \) minimizes the function

\[ \varphi_k(t) = f(x_k - t\nabla f(x_k)) \]  

(1-14)

By the middle of the twentieth century, high-speed computers made the implementation of complex optimization procedures possible and stimulated further research into newer methods. Spectacular advances followed, producing a massive amount of literature on optimization techniques.

**Linear Programming**

Linear programming is defined as the problem of maximizing or minimizing a linear function subject to linear constraints, which may be equalities or inequalities. Here is a simple example.

Find number $x_1$ and $x_2$ that maximize the sum $x_1 + x_2$ subject to the constraints $x_1 \geq 0, x_2 \geq 0$, and

\[
-2x_1 + x_2 \leq 6 \quad (1-15)
\]

\[
x_1 - 3x_2 \leq 2 \quad (1-16)
\]

\[
x_1 + 2x_2 \leq 8 \quad (1-17)
\]

In this problem, there are two unknowns and five constraints. All the constraints are inequalities, and they are all linear in the sense that each involves an inequality in some linear function of the variables.

Since there are only two variables, we can solve this problem by graphing the set of points in the plane that satisfies all the constraints, called the constraint set, and then finding which point of this set maximizes the value of the objective function.

**Dynamic Programming**

The enunciation of the principle of optimality for dynamic programing problems in 1957 by Bellman is known as the *Bellman Equation* [6,7]. The basic idea of the theory of dynamic programming is that of viewing an optimal policy as one that determines the decision required at each moment in terms of the current state of the system. Following this, the basic functional equations given below that describe the quantitative aspects of the theory are uniformly obtained from the following intuitive.
The principle of optimality is an optimal policy having the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

In the case of a discrete deterministic process, any time an M-dimensional vector \( p = (p_1, p_2, ..., p_M) \) is constrained to lie within some region \( D \). Let \( T = \{T_k\} \), where \( k \) runs over a set that may be finite, enumerable, or continuous, and be a set of transformations with the property that \( p \in D \) implies that \( T_k(p) \in D \) for all \( k \).

Let us assume that we are considering an N-stage process to be carried out to maximize some scalar function, \( R(p) \) of the final state. We shall call this function the N-stage return. A policy consists of a selection of \( N \) transformations, \( = (T_1, T_2, ..., T_N) \), successively yielding the state

\[
p_1 = T_1(p)
\]

\[
p_2 = T_2(p_1)
\]

\[
\vdots
\]

\[
p_N = T_N(p_{N-1}) \tag{1-18}
\]

If \( D \) is a finite region, \( T_k(p) \) is continuous in \( p \), and \( R(p) \) is a continuous function of \( p \) for \( p \in D \), it is clear that an optimal policy exists. The maximum value of \( (p_N) \), determined by an optimal policy, will be a function only of the initial vector \( p \) and the number of stages \( N \). Let us then define

\[
f_N(p) = \max_P R(p_N) \tag{1-19}
\]

\( f_N(p) \) = N-stage return obtained using an optimal policy starting from the initial state \( p \).

To derive a functional equation for \( f_N(p) \), we employ the principle cited above. Assume that we choose some transformation \( T_k \) as a result of our first decision, thereby obtaining a new state \( T_k(p) \). The
maximum return from the following \((N - 1)\) stages is, by definition, \(f_{N-1}(T_k(p))\). It follows that \(k\) must now be chosen so as to maximize this value. The result is the basic functional equation

\[
f_N(p) = \max_k f_{N-1}(T_k(p))
\]

\[N = 2, 3, \ldots\]

It is clear that a knowledge of any particular optimal policy, not necessarily unique, will yield \(\{f_N(p)\}\) and that all optimal policies may be determined. We thus have a duality between the space of functions and the space of policies that is of great theoretical and computational importance.

In the discrete stochastic case, let us consider that the transformations are stochastic rather than deterministic. A choice of a transformation \(T_k\) now yields a stochastic vector \(z\) as the new state vector with an associated vector distribution function \(dG_k(p, z)\).

It is clear that it is now generally meaningless to speak of maximizing the return. We must instead agree to measure the value of a policy in terms of some average value of the function of the final state. Let us call this expected value the \(N\)-stage return.

We now define \(f_n(p)\) as before in terms of the \(N\)-stage return. If \(z\) is the state resulting from any initial transformation \(T_k\), the return from the last \((N - 1)\) stages will be \(f_{N-1}(z)\). The expected return as a result of the choice of \(T_k\) is

\[
\int_{z \in D} f_{N-1}(z) dG_k(p, z)
\]

Hence, the functional equation for \(f_N(p)\) is

\[
f_N(p) = \max_k \int_{z \in D} f_{N-1}(z) dG(p, z)
\]

Note that the deterministic process may be considered to be merely a particular case of a stochastic process
If we consider a continuous process in which a decision must be made at each point of a time interval, we are led to maximization problems over function spaces. The simplest examples of these problems are furnished by the calculus of variations as shown below.

Defining

\[ f(p; T) = \text{the return obtained over a time interval } 0, T \]  \hspace{1cm} (1-23)

*using an optimal policy starting from an initial state p*

The analogue of this function is

\[ f(p; S + T) = \max_{D[0,S]} f(T_S(p); T) \]  \hspace{1cm} (1-24)

where the maximum is taken over all allowable decisions made over the initial interval \([0, S]\)

**Non-Linear Programming**

Work by Kuhn and Tucker in 1951 [8] on the necessary and sufficient conditions for the optimal solution of programming problems laid the foundation for later research in non-linear programming. The Kuhn-Tucker conditions provide a unified treatment of constrained optimization in which

- There may be any number of constraints.
- Constraints may be binding or not at the solution.
- Boundary solutions are permitted.
- Non-negativity and structural constraints are treated in the same way.
- Dual variables are shadow values (i.e., marginal values).

The Kuhn-Tucker conditions are first-order necessary conditions for a solution in nonlinear programming to be optimal, provided that some regularity conditions are satisfied.
Let $f : \mathbb{R}^n \to \mathbb{R}$ and $G : \mathbb{R}^n \to \mathbb{R}$ be continuously differentiable functions, and let $b \in \mathbb{R}^n$. We want to characterize those vectors $\hat{x} \in \mathbb{R}^n$ that satisfy

\[ \hat{x} \text{ is solution of the problem } \left\{ \begin{array}{l} \text{Maximize } f(x) \text{ subject to } x \geq 0 \text{ and } G(x) \leq b \\ \text{i.e., subject to } x_1, x_2, \ldots, x_n \geq 0 \text{ and to } G^i(x) \leq b_i \text{ for } i = 1, \ldots, m. \end{array} \right\} \tag{1-25} \]

The Kuhn-Tucker conditions are the first-order conditions that characterize the vector $\hat{x}$ that satisfy (1-25) (when appropriate second-order conditions are satisfied)

\[ \exists \lambda_1, \ldots, \lambda_m \in \mathbb{R}^+ \text{ such that } \left\{ \begin{array}{l} \text{For } j = 1, \ldots, n : \frac{\partial f}{\partial x_j} \leq \sum_{i=1}^m \lambda_i \frac{\partial G^i}{\partial x_j}, \text{with equality if } \hat{x}_j > 0 \\ \text{For } i = 1, \ldots, m : G^i(\hat{x}) \leq b_i, \text{with equality if } \lambda_i > 0 \end{array} \right\} \tag{1-26} \]

where the partial derivatives are evaluated at $\hat{x}$.

The Kuhn-Tucker Conditions given above are in partial derivative form. An equivalent statement of the condition is in gradient form:

\[ \exists \lambda \in \mathbb{R}^m_+ \text{ such that } \left\{ \begin{array}{l} \nabla f \leq \sum_{i=1}^m \lambda_i \nabla G^i \text{ and } \hat{x} \cdot (\nabla f - \sum_{i=1}^m \lambda_i \nabla G^i) = 0 \\ G(\hat{x}) \leq b \text{ and } \lambda \cdot (b - G(\hat{x})) = 0 \end{array} \right\} \tag{1-27} \]

where gradients are evaluated at $\hat{x}$.

**Unconstrained Optimization**

The work of Carroll and Fiacco and McCormick [9,10,11] facilitated the solutions of many difficult problems by using the well-known techniques of unconstrained optimization. The Sequential Unconstrained Minimization Technique (SUMT) is considered to be simplest of these techniques to implement. The main idea is to solve a sequence of unconstrained problems that differ only in the value
of the resistance factor $r$. This factor will gradually be reduced, which has the effect of weakening the repulsive effect of the barrier function for the inequalities and strengthening its attractive effect for the equality constraints.

In this technique, the constrained, concave programming problem, maximize $f(x)$, subject to $g_i(x) \geq 0, i = 1, 2, ..., m$, and $x$ is an $n$-dimensional vector, is modified to the unconstrained problem;

$$\text{Maximize } P(x, \tau) = f(x) - \tau \sum_{i=1}^{m} \frac{w_i}{g_i(x)}$$

(1-28)

with $w_i \geq 0$ and $\tau > 0$

The unconstrained $P$ function has several desirable properties;

1) The summation term, called the penalty function, approaches $-\infty$ if the boundary of a constraint is reached.

2) If the functions $f(x)$ and $g_i(x)$ have continuous first and second partial derivatives inside the feasible region, then the well-known necessary conditions for an unconstrained function to have a local maximum apply. That is, the gradient vanishes at that point and the matrix of second partials is negative semi-definite.

3) If $f(x)$ is strictly concave or any $g_i(x)$ is strictly concave then the local maximum is the global maximum.

4) First- or second-order gradient methods may be used to maximize $P(x, \tau)$

5) When a maximum of $P(x, \tau)$ is reached for a fixed value of $\tau$, $\tau$ can be reduced ($\tau_1 > \tau_2 > \cdots \tau_p > 0$) and the new $P(x, \tau)$ solved for a maximum.

6) As $\tau_p \to 0$, the maximum of $P(x, \tau_p)$ approaches the maximum of $f(x)$, and the penalty term approaches zero.
Carroll [10] demonstrated that the computational method will solve convex programming problems. Fiacco and McCormick [11] proved Carroll’s conjecture for the corresponding convex programming problem. Their work established the proof of the following:

Given the convex programming problem

Minimize $f(x)$

subject to $g_i(x) \geq 0, i = 1,2, \ldots, m$ (where $x$ is a n-dimensional vector)

Define the function:

$$P(x, \tau) = f(x) + \tau P \sum_{i=1}^{m} \frac{1}{g_i(x)}$$

where $\tau_1 > \tau_2 > \ldots \tau_p > 0$

Define $x(\tau)$ as the vector minimizing $P(x, \tau_p)$, and let $V_o$ be the constrained minimum value of $f(x)$.

Then:

$$\lim_{p \to \infty, \tau_p \to 0} P(x, \tau_p) = V_o$$

The following additional condition must hold.

1) $R^+ = \{x | g_i(x) > 0, i = 1, \ldots, m\}$ is not empty, denote by $R$ the closure of $R^+$.

2) $f(x)$ and $-g_i(x)$ are convex and twice continuously differentiable for $x \in R$.

3) For every finite $h, \{x | f(x) \leq h, x \in R\}$ is a bounded set.

4) For every $\tau > 0, P(x, \tau)$ is strictly convex.
Geometric Programming

Geometric programming (GP) was developed in the 1960s by Duffin, Zener, and Peterson [12]. GP is an optimization problem of the form

\[
\text{Minimize } f_0(x) \quad (1-30)
\]

subject to \( f_i(x) \leq 1, i = 1, \ldots, m, \) and \( g_i(x) = 1, i = 1, \ldots, p, \)

where \( f_i \) are posynomial functions, \( g_i \) are monomials, and \( x_i \) are the optimization variables.

A posynomial is a function of the form

\[
f(x_1, x_2, \ldots, x_n) = \sum_{K=1}^{K} c_k x_1^{a_{1k}} \cdots x_n^{a_{nk}} \quad (1-31)
\]

where all the conditions and coefficients are positive real number, and exponents are real numbers.

Posynomial are closed under addition, multiplication, and nonnegative scaling. (There is an implicit constraint that the variables are positive, i.e., \( x_i > 0 \).) In a standard form GP, the objective must be posynomial and must be minimized. The equality constraints can only have the form of a monomial equal to one, and the inequality constraints can only have the form of a posynomial less than or equal to one.

Geometric programs are not convex optimization problems, but can be transformed to convex problems by a change of variables and a transformation of the objective and constraint functions. In particular, defining \( y_i = \log(x_i) \), the monomial

\[
f(x) = c x_1^{a_1} \cdots x_n^{a_n} \rightarrow e^{a^T y + b} \quad (1-32)
\]

where \( b = \log(c) \). Similarly, if \( f \) is the posynomial, then

\[
f(x) = \sum_{k=1}^{K} e^{a_k^T y + b_k} \quad (1-33)
\]
where \( a_k = (a_{1k}, ..., a_{nk}) \) and \( b_k = \log(c_k) \). After the change of variables, a posynomial becomes a sum of exponentials of affine functions.

**Integer Programming**

One of the most exciting and rapidly developing areas of optimization. Integer linear programming (ILP) differs from linear programming in the basic context of feasible region. In LP, the feasible region is basically a convex set in some finite dimension, but in an ILP, the feasible sets are nothing but disjointed unions of discrete points. Here is a graphical representation:

![Graphical representation of LP and ILP feasible regions](image)

Figure 1-1: Feasible regions of LP and ILP. The black circles denote the ILP solutions, while the shaded region represents the feasible region of the LP.

The cutting planes method was proposed by Ralph Gomory [13] in the 1950s as a method for solving the ILP. It was considered impractical due to numerical instability, however, in 1990s, Gerard Cornuejols [14] showed the cutting planes methods to be effective in combination with branch-and-bound and a way to overcome numerical instabilities.
The ILP problem can be formulated as

\[
\text{Maximize } c^T x
\]

subject to \( Ax = b, x \geq 0, \text{ and } x_i \text{ all integers.} \)

Then, drop the requirement that the \( x_i \) be integers and solve the associated linear problem to obtain a basic feasible solution. Geometrically, this solution will be a vertex of the convex polytope consisting of all feasible points. If the vertex is not an integer point, then the method finds a hyperplane with the vertex on one side and all feasible integer points on the other. This is then added as an additional linear constraint to exclude the vertex found, creating a modified linear program. The new program is solved and the process is repeated until an integer solution is found.

The necessity to optimize more than one objective or goal while satisfying the physical limitations led to the development of multi-objective programming methods. Goal programming is a well-known technique for solving specific types of multi-objective optimization problems and was originally proposed for linear problems by Charnes and Cooper [15,16] in the 1960s.

The basic approach of goal programming is to establish a specific numeric goal for each of the objectives, formulate an objective function for each objective, and then seek a solution that minimizes the sum of deviations of these objective functions from their respective goals. There are three possible types of goals:

1) A lower, one-sided goal that sets a lower limit that we do not want to fall under
2) An upper, one-sided goal that sets an upper limit that we do not want to exceed
3) A two-sided goal that sets a specific target that we do not want to miss on either side.

A major strength of goal programming is its simplicity and ease of use, therefore, goal programming can handle relatively large numbers of variables, constraints, and objectives.
The foundation of game theory was laid by von Neumann [17,18] in 1928 and since then the technique has been applied to solve several mathematical, economic and military problems. Its weakness is the ability to produce solutions that are not Pareto efficient. This violates a fundamental concept of decision theory that no rational decision maker will knowingly choose a solution that is not Pareto efficient.

Metaheuristic Method

Metaheuristic methods such as simulated annealing, neural network methods, and nature-inspired optimization methods represent a new class of mathematical programming techniques that came into prominence during the 1990s.

Simulated annealing was proposed by Kirkpatrick et al. [19], and mimics the physical process of annealing metals and glass. It tries to improve a solution by walking randomly in the space of possible solutions and gradually adjusting a parameter called temperature. At high temperature, the random walk is almost unbiased and essentially converges to a uniform distribution over the whole space of solutions with a better objective value, with the distribution more and more biased towards the optimal solutions. The sequence of temperatures and lengths of time for which they are maintained is called the annealing schedule in analogy with statistical mechanics.

Simulated annealing is a special case of a stochastic search method that starts with one distribution and gradually changes it to another target distribution. The intermediate distribution satisfies the following two properties.

1) Any two consecutive distributions must not be too different; one way to formalize this is to require that their total variation distance is bounded away from 1.
2) All the distribution along the way must be efficiently sample-able. The most general class of distributions for which we currently have efficient sampling class of log-concave distributions.

![Diagram](image1)

Figure 1-2: Optimization over a high-dimensional cone of a pair of consecutive Boltzman distribution $e^{-cx/T}$. The standard derivation $\sigma$ in the direction of optimization is large.

![Diagram](image2)

Figure 1-3: Optimization over a high-dimensional cone of a pair of uniform distribution over truncated cones $c.x \leq T$. The standard derivations are much smaller, allowing less movement and requiring more phases.

In simulated annealing, the intermediate distributions are all from the exponential family, that has density at $x$ is proportional to $e^{-c^Tx}$ for some vector $c$, restricted to some domain.
In 1943, Warren McCulloch and Walter Pitts [20] created a computational model for neural networks based on mathematics and algorithms called *threshold logic*. This model paved the way for neural network research. Neural network methods are based on solving problems using the computing power of a network of interconnected “neuron” processors. In the late 1940s psychologist Donald Hebb created a hypothesis of learning based on the mechanism of neural plasticity that is known as *Hebbian learning* [21]. It is considered to be a typical unsupervised learning rule and its later variants were early models for long term potentiation. Farley and Clark [22] at MIT first used computational machines called *calculators* to simulate a Hebbian network. Other neural network computational machines were created by Rochester, Holland, Habit, and Duda [23].

The very simplest networks contain no hidden layers and are equivalent to a linear regression. Figure 1-4 shows the neural network version of a linear regression with four predictors. The forecasts are obtained by a linear combination of the inputs. The weights are selected in the neural network framework using a *learning algorithm* that minimizes a *cost function*.

![Neural Network Diagram](image)

Figure 1-4: A simple neural network equivalent to a linear regression.
Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear.

A simple example is shown in Figure 1-5.

![Figure 1-5: A neural network with four inputs and one hidden layer with three hidden neurons.](image)

To utilize neural networks, a method of global optimization must be found. The most frequently used algorithm for optimizing a neural network is back propagation such as a gradient search technique that is likely to obtain local solutions. Other methods include simulated annealing, which performs better because it is a global search algorithm. Many more researchers have tried to utilize other global search algorithms such as the nature-inspired optimization methods e.g., the genetic algorithm.
1.2 Brief History of Nature-Inspired Optimization Algorithms

The genetic algorithm (GA) was developed in the 1960s by John Holland and his collaborators [24] at the University of Michigan. He studied the adaptive system and was the first to use crossover and recombination manipulations for modeling such a system. His book summarizing the development of genetic algorithms was published in 1975. Genetic algorithms appear to be able to systematically obtain superior solutions compared to simulated annealing for optimizing neural networks. These solutions provide the researcher with superior estimates of interpolation data.

GA is a search method based on the abstraction of Darwinian evolution and the natural selection process of a biological system and represents these factors in the mathematical operators: crossover, recombination, mutation, and fitness. Since its development, GA has been successful in solving a wide range of optimization problems.

In 1992, Marco Dorigo [25] finished his Ph.D. thesis on optimization and natural algorithms, in which he described his innovative work on ant colony optimization (ACO). ACO is a search technique that was inspired by the swarm intelligence of social ants using pheromones as a chemical messenger. In 1995, the particle swarm optimization (PSO) algorithm was introduced by James Kennedy and Russell C. Eberhart [26] and gained popularity over other evolutionary algorithms because of its simplicity in implementation and efficient optimization. PSO is inspired by the swarm intelligence of fish and birds with multiple agents, called particles, swarming around the search space starting from some initial random guess. The swarm communicates the current best and shares the global best. In 2008, Xin-She Yang [27] introduced the firefly algorithm (FA) in his book, in which, he proposed a new search method inspired by the behavior of fireflies in flashing their lights. The attraction of fireflies to brightness is used to bias the movement of fireflies in order to converge on the optimal point. These four nature-inspired optimization methods will be discussed in more detail in Chapter 2.
1.3 Motivation

Optimization techniques are essential in our daily lives in engineering and industry and are applied in almost all fields and applications. Optimization is imperative in the real world because of limitations in resources, time, and money. Classical optimization problem-solving methodologies involve exact methods and heuristic methods. The heuristic approach seems to be superior in solving difficult and complex optimization problems. Nature-inspired optimization methods such as GA, PSO, ACO, and FA are among the most used and latest metaheuristic methods, which make use of many random decisions. A vast literature exists on nature-inspired approaches for solving an array of problems and, more recently, a number of studies have reported on the success of such techniques for solving difficult problems. Still, not many studies apply these optimization techniques for problems in antenna impedance matching network design.

The first goal of this dissertation is to evaluate the performance of four nature-inspired optimization methods on one of antenna design problem, the impedance matching problem. By varying the parameter settings, we try to find their best settings for this global optimization problem. First, we compare them with MATLAB optimization tool solvers and with each other. The objective function is to minimize the maximum VSWR over a bandwidth.

The second goal of this dissertation is to enhance the approach that shows very good results, the FA, to achieve better results. The newly modified firefly algorithm (MFA) is tested on eleven testing functions. The final problems for this dissertation are the impedance matching problem including both lossless impedance matching networks and lossy impedance matching networks. The results of the new MFA and standard FA were compared.
1.4 Organization of the Dissertation

Chapter 1 provides an introduction to and history of optimization algorithms, including brief explanations of some of the algorithms and how they contribute to the development of optimization research. Chapter 2 details four nature-inspired optimization algorithms. GA, PSO, ACO, and FA are presented and their standard algorithms, which are used in this dissertation, are explained in detail.

Chapter 3 provides the basic concept of the impedance matching problem and its solutions, including both lossless and lossy networks. Chapter 4 shows the results from the optimization of a high frequency dipole antenna using four nature-inspired optimization algorithms and compares those results with the result when using the global optimization toolbox from the MATLAB software.

Chapter 5 begins by describing the standard firefly algorithm (SFA) in detail and then explains how the new modified firefly algorithm (MFA) is different from SFA. Both SFA and MFA were used to find the optimal solution for six test functions, and their results were compared and analyzed. Chapter 6 uses the impedance matching network as the objective problem for SFA and MFA.

Chapter 7 presents conclusion and some suggestions for future research.
Chapter 2

Nature-Inspired Optimization Methods

The nature-inspired optimization methods inspiration comes from the behavior of organisms and naturally occurring processes. These methods have the ability to describe and resolve the complex relationship from intrinsically very simple initial conditions and rules with little or no knowledge of the search space. Nature is the perfect example for optimization, because if we closely examine every feature or phenomenon in nature, it always finds the optimal strategy, while addressing complex iterations among organisms ranging from microorganisms to full-fledged human beings, maintaining diversity and balancing the ecosystem, adaptation, physical phenomena like river formation, forest fires, clouds, rains, etc. Even though the strategy behind the solution is simple the results are amazing. Nature is truly the best teacher, and its designs and capabilities are both enormous and mysterious. Now that researchers are trying to mimic nature in technology, the two fields have a much stronger connection since, it seems entirely reasonable that new or persistent problems in science could have a great deal in common with problems nature encountered and resolved long ago. Thus, an easy mapping is possible between nature and technology.
2.1 Introduction of Nature-Inspired Optimization Methods

Nature-inspired computing has developed as a new era in computing encompassing a wide range of applications, covering almost all areas including computer networks, security, robotics, bio-medical, control systems, power systems, and many more.

One of the approaches is to use metaheuristic algorithms that can find an answer to a problem by implementing a form of stochastic optimization method, which randomly searches for the optimal solution [27]. Because of their randomness, these methods are less sensitive to modeling errors, and enable the search to escape a local minimum and eventually to approach a global optimum. In short, they are a simple and effective way to obtain algorithms with good performance for all kind of problems. Some examples of stochastic optimization are simulated annealing, swarm intelligence algorithms and evolutionary algorithms.

Swarm intelligence algorithms are derived from the social behavior of animals that consist of a group of non-intelligent simple agents with no central-control or self-control to their behavior [28]. The local interaction between the agents and their environment lead to the swarm behavior that is new to the individual agents. Examples of swarm intelligence algorithms are particle swarm optimization (PSO), ant colony optimization (ACO), and firefly algorithm (FA).
2.2 Genetic Algorithm (GA)

2.2.1 History of Genetic Algorithm (GA)

In the 1950s several computer scientists independently studied evolutionary systems with the idea that evolution could be used as an optimization tool for engineering problems. The purpose of these systems was to evolve a population of candidate solutions to a given problem, using operators inspired by natural genetic variation and natural selection.

In the 1960s and 70s, Rechenberg [29,30] introduced "evolution strategies", a method he used to optimize real valued parameters for devices such as airfoils. This idea was further developed by Schwefel [31,32]. The field of evolution strategies has remained an active area of research, mostly developing independently from the field of genetic algorithms. In 1966, Fogel, Owens, and Walsh [33] developed "evolutionary programming," a technique in which candidate solutions to given tasks were represented as finite state machines, which were evolved by randomly mutating their state transition diagrams and selecting the fittest. A broader formulation of evolutionary programming also remains an area of active research [29]. Together, evolution strategies, evolutionary programming, and genetic algorithms form the backbone of the field of evolutionary computation.

Several other people working in the 1950s and 1960s [34,35,36,37,38] developed evolution–inspired algorithms for optimization and machine learning, although, their work has been given little or none of attention or follow up that evolution strategies, evolutionary programming, and genetic algorithms have seen. In addition, a number of evolutionary biologists used computers to simulate evolution for the purpose of controlled experiments [39,40,41,42,43].

Genetic algorithms (GAs) were invented by John Holland [24] in the 1960s and were developed by Holland and his students and colleagues at the University of Michigan in the 1960s and 1970s. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to
design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Holland presented the GA as an abstraction of natural evolution and gave a theoretical framework for adaptation under the GA.

Holland's GA is a method for moving from one population of "chromosomes" to a new population by using a kind of "natural selection" together with the genetics–inspired operators of crossover, mutation, and inversion. Each chromosome consists of "genes" e.g., bits, each gene being an instance of a particular "allele" e.g., 0 or 1. The selection operator chooses those chromosomes in the population that will be allowed to reproduce, and on average the fitter chromosomes produce more offspring than the less fit ones. The crossover exchanges subparts of two chromosomes, roughly mimicking biological recombination between two single chromosome organisms; mutation randomly changes the allele values of some locations in the chromosome; and inversion reverses the order of a contiguous section of the chromosome, thus rearranging the order in which genes are arrayed.

Rechenberg's evolution strategies [29,30] started with a "population" of two individuals, one parent and one offspring, the offspring being a mutated version of the parent; many individual populations and crossover were not incorporated until later. Fogel, Owens, and Walsh's evolutionary programming [33] likewise used only mutation to provide variation. Holland's introduction of a population–based algorithm with crossover, inversion, and mutation was a major innovation. Moreover, Holland was the first to attempt to put computational evolution on a firm theoretical footing [24]. Until recently this theoretical foundation, based on the notion of "schemas," was the basis of almost all subsequent theoretical work on genetic algorithms.
2.2.2 Search Spaces and Elements

The idea of searching among a collection of candidate solutions for a desired solution is so common in computer science. The term search space refers to some collection of candidate solutions to a problem and some notion of distance between candidate solutions. For example, considering the problem of computational protein design, we want to use a computer to search for a protein a sequence of amino acids that folds up into a particular three-dimensional shape so it can be used to fight a specific virus. The search space is the collection of all possible protein sequences an infinite set of possibilities. To constrain it, let us restrict the search to all possible sequences having a length 100 or less still a huge search space, since there are 20 possible amino acids at each position in the sequence. If we represent the 20 amino acids by using letters of the alphabet, the candidate solutions will look like this.

\[ B C C M C G D L \ldots \]

We define the distance between two sequences as the number of positions in which the letters at corresponding positions differ. For example, the distance between \( B C C M C G D L \) and \( G C C M C G D L \) is 1, and the distance between \( B C C M C G D L \) and \( L B M P A G G A \) is 7. An algorithm for searching this space is a method for choosing which candidate solutions to test at each stage of the search. In most cases, the next candidate solutions to be tested will depend on the results of testing previous sequences; most useful algorithms assume that there will be some correlation between the quality of neighboring candidate solutions those close in the space. Genetic algorithms assume that high quality parent candidate solutions from different regions in the space can be combined via crossover to, on occasion, produce high-quality offspring candidate solutions.

Another important concept is that of a fitness. Originally defined by the biologist [45] in the context of population genetics, a fitness landscape is a representation of the space of all possible genotypes along with their fitness.
The idea of evolution moving populations around in unchanging landscapes is biologically unrealistic for several reasons. For example, an organism cannot be assigned a fitness value independent of the other organisms in its environment; thus, as the population changes, the fitness of particular genotypes will change as well. In other words, in the real world the "landscape" cannot be separated from the organisms that inhabit it.

One common application of GAs is function optimization, where the goal is to find a set of parameter values that maximize, say, a complex multi parameter function. As a simple example, one might want to maximize the real valued one-dimensional function

\[ f(y) = y + |\sin 32y|, \quad 0 \leq y \leq \pi \] (2-1)

The candidate solutions are values of \( y \), which can be encoded as bit strings representing real numbers. The fitness calculation translates a given bit string \( x \) into a real number \( y \) and then evaluates the function at that value. The fitness of a string is its function value at that point.

As a non-numerical example, consider the problem of finding a sequence of 50 amino acids that will fold to a desired three-dimensional protein structure. A GA could be applied to this problem by searching a population of candidate solutions, each encoded as a 50-letter string such as

\[ IHASDCVSAAMJRVTVASYLKNWTSMNWEEICIOPLKWDDNFLKM, \]

where each letter represents one of 20 possible amino acids. One way to define the fitness of a candidate sequence is as the negative of the potential energy of the sequence with respect to the desired structure. The potential energy is a measure of how much physical resistance the sequence would put up if forced to be folded into the desired structure the lower the potential energy, the higher the fitness. One would not want to physically force every sequence in the population into the desired structure and measure its resistance; this would be very difficult, if not impossible. Instead, given a sequence and a
desired structure, one can estimate the potential energy by calculating some of the forces acting on each amino acid, so the whole fitness calculation can be done computationally.

These examples show two different contexts in which candidate solutions to a problem are encoded as abstract chromosomes encoded as strings of symbols, with fitness functions defined on the resulting space of strings. A GA is a method for searching such fitness landscapes for highly fit strings. The simplest form of GA involves three types of operators: selection, crossover, and mutation.

1) Selection: This operator selects chromosomes in the population for reproduction. The fitter the chromosome, the more times it is likely to be selected to reproduce.

2) Crossover: This operator randomly chooses a locus and exchanges the subsequences before and after that locus between two chromosomes to create two offspring. For example, the strings 00010100 and 11111110 could be crossed over after the third locus in each to produce the two offspring 10011110 and 01110100. The crossover operator roughly mimics biological recombination between two single chromosome organisms.

3) Mutation This operator randomly flips some of the bits in a chromosome. For example, the string 10011110 might be mutated in its second position to yield 00011110. Mutation can occur at each bit position in a string with some probability, usually very small (e.g., 0.001).

2.2.3 Simple Genetic Algorithm (SGA)

Given a problem to be solved, a simple GA works as follows,

1) An initial population of $N_{pop}$ chromosomes is constructed randomly.

2) The value of elements is calculated, and the fitness of the population of chromosomes is evaluated in proportion to the objective value.

3) A selection process follows, during which a new population of the size $N_{pop}$ is created. During selection, individuals are chosen from the old population with a probability equal to the ratio of
the individuals’ fitness to the total fitness of the old population. Regardless of the result, the individuals with the $N_{\text{elite}}$ highest fitness values are selected to guarantee their genetic information will be passed along to the next generation.

4) After selection, individuals are paired off as parents and cross over $N_{\text{cross}}$ times, each with probability $p_{\text{cross}}$. Each time crossover occurs, the chromosomes are split at a random bit location and their partial bit strings are swapped between the parents.

5) Following crossover, a mutation is performed by negating each bit on each chromosome with probability $p_{\text{mut}}$. The fitness of the resulting population is evaluated, and the process is repeated for $N_{\text{gen}}$ generations.

6) Repeat 3) 4) and 5) until $N_{\text{pop}}$ offspring have been created

7) Replace the current population with the new population and go to step 2)

Each iteration of this process is called a generation. A GA is typically iterated for anywhere from 50 to 500 or more generations. The entire set of generations is called a run. At the end of a run there are often one or more highly fit chromosomes in the population. Since randomness plays a large role in each run, two runs with different random–number seeds will generally produce different detailed behaviors. GA researchers often report statistics (such as the best fitness found in a run and the generation at which the individual with that best fitness was discovered) averaged over many different runs of the GA on the same problem.

The simple procedure just described is the basis for most applications of GAs. There are a number of details to fill in, such as the size of the population and the probabilities of crossover and mutation, and the success of the algorithm often depends greatly on these details. There are also more complicated versions of GAs e.g., GAs that work on representations other than strings or GAs that have different types of crossover and mutation operators.
### Figure 2-1: Pseudo-code of Genetic Algorithm (GA).

```plaintext
// Initialize generation = 0,

k = 0;

P_k = a population of n randomly-generated individuals;

// Evaluate P_k:

Compute fitness(i) for each i ∈ P_k;

Do
{
    // Create generation k + 1:

    // 1. Copy:

    Select (1 - χ) × n members of P_k and insert into P_{k+1};

    // 2. Crossover:

    Select χ × n members of P_k; pair them up; produce offspring; insert the offspring into P_{k+1};

    // 3. Mutate:

    Select μ × n members of P_{k+1}; invert a randomly-selected bit in each;

    // Evaluate P_{k+1}:

    Compute fitness(i) for each i ∈ P_k;

    // Increment:

    k = k + 1;
}

while fitness of fittest individual in P_k is not high enough;

return the fittest individual from P_k;
```
2.3 Particle Swarm Optimization (PSO)

2.3.1 History of Particle Swarm Optimization (PSO)

In the mid-1990s, Kennedy and Eberhart [46] introduced an alternative solution to the complex non-linear optimization problem by emulating the collective behavior of bird flocks simulated as particles, and called it particle swarm optimization (PSO).

This algorithm does not require any gradient information of the function to be optimized and uses only primitive mathematical operators, and is conceptually very simple. PSO can be implemented very easily in any computer language and requires minimal parameter tuning. Also, the performance of the algorithm does not deteriorate severely with the growth of the search space dimensions.

2.3.2 Classical PSO

Eberhart’s PSO approach [47] involves defining the global optimum of an n-dimensional function that is to be located. The function may be represented as:

\[ f(x_1, x_2, \ldots, x_n) = f(\vec{X}) \]  

where \( \vec{x} \) is the search variable vector, which represents the set of independent variables of a given function. The task is to find out such a \( \vec{x} \) that the function value \( f(\vec{x}) \) is either a minimum or a maximum denoted by \( f^* \) in the search range. If the components of \( \vec{x} \) assume real values, then the task is to locate a particular point in the \( n \)-dimensional hyperspace which is a continuum of such points.

PSO is a multi-agent parallel search technique. Particles are conceptual entities, that fly through the multi-dimensional search space. At any particular instant, each particle has a position and a velocity. The position vector of a particle with respect to the origin of the search space represents a trial solution of the search problem.

At the beginning, a population of particles is initialized with random positions marked by the vectors \( \vec{x}_i \) and random velocities \( \vec{v}_i \). The population of such particles is called a “swarm” \( S \). A
neighborhood relation $N$ is defined in the swarm. $N$ determines for any two particles $P_i$ and $P_j$ whether they are neighbors or not. Thus, for any particle $P$, a neighborhood can be assigned as $N(P)$, containing all the neighbors of that particle. Different neighborhood topologies and their effects on swarm performance will be discussed later. However, a popular version of PSO uses $N = S$ for each particle. In this case, any particle has all the remaining particles in the swarm in its neighborhood.

Each particle $P$ has two state variables viz., its current position $\vec{x}(t)$ and current velocity $\vec{v}(t)$. Each particle is also equipped with a small memory comprising its previous best position (the one yielding the highest value of the fitness function found so far) $\vec{p}(t)$, i.e., the personal best experience and the best $\vec{p}(t)$ of all $\in N(P)$: $\vec{g}(t)$ , i.e., the best position found so far in the neighborhood of the particle. When we set $N(P) = S$, $\vec{g}(t)$ is referred to as the globally best particle in the entire swarm.

The PSO scheme has the following algorithmic parameters:

1) $V_{max}$ is a maximum velocity that restrict $\vec{V}_i(t)$ within the interval $[-V_{max}, V_{max}]$.
2) An inertial weight factor $\omega$.
3) Two uniformly distributed random numbers $\varphi_1$ and $\varphi_2$ that respectively determine the influence of $\vec{p}(t)$ and $\vec{g}(t)$ on the velocity update formula.
4) Two constant multiplier terms $C_1$ and $C_2$ known as “self-confidence” and “swarm confidence”, respectively.

Initially the settings for $\vec{p}(t)$ and $\vec{g}(t)$ are $\vec{p}(0) = \vec{g}(0) = \vec{x}(0)$ for all particles. Once the particles are all initialized, an iterative optimization process begins, where the positions and velocities of all the particles are altered by the following recursive equations. The equations are presented for the $d$th dimension of the position and velocity of the $i$th particle.

$$V_{id}(t + 1) = \omega \cdot v_{id} + C_1 \cdot \varphi_1 \cdot (P_{id}(t) - x_{id}(t)) + C_2 \cdot \varphi_2 \cdot (g_{id}(t) - x_{id}(t)) \quad (2-3)$$

$$x_{id}(t + 1) = x_{id}(t) + v_{id}(t + 1) \quad (2-4)$$
The first term in the velocity updating formula represents the inertial velocity of the particle, 
\( \omega \), and is called the inertia factor. Venter and Sobeiski [48] termed \( C_1 \) as “self-confidence” and \( C_2 \) as “swarm confidence”. These terminologies provide an insight from a sociological standpoint. Since the coefficient \( C_1 \) has a contribution towards the self-exploration of a particle, we regard it as the particle’s self confidence. On the other hand, the coefficient \( C_2 \) has a contribution towards the motion of the particles in a global direction, which takes into account the motion of all the particles in the preceding program iterations, so naturally its definition as “swarm confidence” is apparent. \( \varphi_1 \) and \( \varphi_2 \) stand for uniformly distributed random numbers in the interval [0,1]. After having calculated the velocities and position for the next time step \( t + 1 \), the first iteration of the algorithm is completed. Typically, this process is iterated for a certain number of time steps, or until some acceptable solution has been found by the algorithm or an upper limit of CPU usage has been reached.

```
For each particle
    Initialize particle
END
Do
    For each particle
        Calculate fitness value
        If the fitness value is better than the best fitness value \( p_{\text{best}} \) in history
            Set current value as the new \( p_{\text{best}} \)
        End
    Choose the particle with the best fitness value of all particles as the \( g_{\text{best}} \)
    For each particle
        Calculate particle velocity according equation
        Update particle position according equation
    End
While maximum iterations or minimum error criteria is not attained
```

Figure 2-2: Pseudo-code Particle Swarm Optimization (PSO).
2.3.3 Selection of Parameters for PSO

The main parameters of the standard PSO model are $\omega$, $C_1$, $C_2$, $V_{max}$ and the swarm size $S$. The settings of these parameters determine how the model optimizes the search-space. For instance, one can apply a general setting that gives reasonable results on most problems but is seldom optimal. Since the same parameter settings do not guarantee success in different problems, we must have knowledge of the effects of the different settings, such that we can pick suitable settings from problem to problem.

The Inertia Weight $\omega$

The inertia weight $\omega$ controls the momentum of the particle. If $\omega \ll 1$, only a little momentum is preserved from the previous time-step, thus, quick changes of direction are possible with this setting. The concept of velocity is completely lost if $\omega = 0$, and the particle then moves in each step without knowledge of the past velocity. On the other hand, if $\omega$ is high, we observe the same effect as when $C_1$ and $C_2$ are low: particles can hardly change their direction and turn around, which of course implies a larger area of exploration as well as a reluctance against convergence towards an optimum. Setting $\omega > 1$ must be done with care, since velocities are further biased for exponential growth. This setting is rarely seen in PSO implementation, and is always used together with $V_{max}$. In short, high settings near 1 facilitate global search, and lower settings in the range $[0.2, 0.5]$ facilitate rapid local search.

Eberhart and Shi have studied $\omega$ in several papers [49,50,51,52] and found that “when $V_{max}$ is not small ($\geq 3$), an inertia weight of 0.8 is a good choice”. Although this statement is solely based on a single test function, this setting is in fact a good choice in many cases. Eberhart and Shi also devised an adaptive fuzzy PSO, where a fuzzy controller was used to control $\omega$ over time [53]. This approach is very interesting, since it potentially lets the PSO self-adapt $\omega$ to the problem and thus optimizes and eliminates a parameter of the algorithm, which saves time during experimentation since fine-tuning of $\omega$ is no longer
necessary anymore. At each time-step, the controller takes the “Normalized Current Best Performance Evaluation” and the current setting of $\omega$ as inputs, and it outputs a probabilistic change in $\omega$.

**The Maximum Velocity $V_{\text{max}}$**

The maximum velocity $V_{\text{max}}$ determines the maximum change one particle can undergo in its positional coordinates during an iteration. Usually we set the full search range of the particle’s position as the $V_{\text{max}}$. For example, in a case, in which a particle has position vector $\vec{x} = (x_1, x_2, x_3)$ and if $-10 \leq x_i \leq 10$ for $i = 1, 2, \text{and} 3$, then we set $V_{\text{max}} = 20$.

**The Swarm Size $S$**

It is quite a common practice in the PSO literature to limit the number of particles to the range of 20-60 [12,18]. Van den Bergh and Engelbrecht [54] have shown that although there is a slight improvement of the optimal value with increasing swarm size, a larger swarm increases the number of function evaluations to converge to an error limit. Eberhart and Shi [51] illustrated that the population size has hardly any effect on the performance of the PSO method.

**The Acceleration Coefficients $C_1$ and $C_2$**

A usual choice for the acceleration coefficients $C_1$ and $C_2$ is $C_1 = C_2 = 1.494$ [55]. However, other settings have also been used in different papers. Usually $C_1$ equals $C_2$ and ranges from [0,4]. Ratnaweera et al. have investigated the effect of varying these coefficients with time in [56] and adapted $C_1$ and $C_2$ with time in the following way:

$$C_1 = (C_{1f} - C_{1i}) \frac{\text{iter}}{\text{MAXITER}} C_{1i}$$

(2-5)

$$C_2 = (C_{2f} - C_{2i}) \frac{\text{iter}}{\text{MAXITER}} C_{2i}$$

(2-6)
Where $C_{1i}, C_{1f}, C_{2i}, and C_{2f}$ are constants, $iter$ is the current iteration number and $MAXITER$ is the maximum number of allowable iterations. The objective of this modification was to boost the global search over the entire search space during the early part of the optimization and to encourage the particles to converge toward a global optimum at the end of the search.

### 2.3.4 Neighborhood Topologies in PSO

The most commonly used PSOs are either the global version or local version. In the global version of PSO, each particle flies through the search space with a velocity that is dynamically adjusted according to the particle’s personal best performance achieved so far and the best performance achieved so far by all the particles. However, in the local version of PSO, each particle’s velocity is adjusted according to its personal best and the best performance achieved so far within its neighborhood. The neighborhood of each particle is generally defined as including the topologically nearest particles to all sides. The global version of PSO also can be considered as a local version of PSO with each particle’s neighborhood treated as the whole population. It has been suggested that the global version of PSO converges quickly, but with the potential to converge to the local minimum, while the local version of PSO might have more chances to find better solutions slowly [26]. Recently, many researchers have worked on improving the performance of global PSO by designing or implementing different types of neighborhood structures. Some [57] claimed that PSOs with small neighborhoods might perform better on complex problems while PSOs with large neighborhood would perform better for simple problems.

The $k$-best topology, proposed by Kennedy, connects every particle to its $k$ nearest particles in the topological space. When $k = 2$, this becomes a circle topology (and with $k = 1$ it becomes a $g_{best}$ topology) also known as the wheel topology, in which the only connections are from one central particle to the others (see Fig. X). In addition, one could imagine a huge number of other topologies.
Fig 2-3: Topologies of Particle Swarm Optimization (PSO) (a) The fully connected topology. (b) k-best nearest neighbor. (c) wheel topology.
2.4 Ant Colony Optimization (ACO)

An ant colony is highly organized, which is possible because each ant interacts with others in the colony through pheromones. Although ant species are almost blind, they can still communicate with the environment and with each other by dropping pheromones on the paths they take. These pheromone trails then act as stimuli for other ants, which are more likely to follow the paths that have relatively high levels of pheromones. Therefore, an ant that has decided to follow a path due to the pheromone trail already on that path reinforces the path further by laying down its own pheromone as well.

The first ant colony optimization (ACO) algorithm, which was introduced by Marco Dorigo, was called the Ant System (AS) [58]. ACO algorithms were inspired by studying the behavior of ants and were originally proposed for combinatorial or discrete problems. ACO algorithms were first implemented in the travelling salesman problem (TSP) [59]. Recently, however, they have also been applied to solve continuous or mixed problems.

2.4.1 Ant System (AS)

AS was the first ACO algorithm proposed. Its main characteristic is that, at each iteration, the pheromone values are updated by all the m ants that have built a solution in the iteration itself. The pheromone $\pi_{ik}$, associated with the edge-joining cities $i$ and $j$, is updated as follows:

$$
\pi_{ij} \leftarrow (1 - \rho) \cdot \pi_{ij} + \sum_{k=1}^{m} \Delta \pi_{ij}^k
$$

where $\rho$ is the evaporation rate, $m$ is the number of ants, and $\Delta \pi_{ij}^k$ is the quantity of pheromone laid on edge $(i, j)$ by ant $k$:

$$
\Delta \pi_{ij}^k = \begin{cases} 
\frac{Q}{L_k} & \text{it ant } k \text{ used edge } (i, j) \text{ in its tour,} \\
0 & \text{otherwise}
\end{cases}
$$
Where $Q$ is a constant, and $L_k$ is the length of the tour constructed by ant $k$

In the construction of a solution, ants select the following city to be visited through a stochastic mechanism. When ant $k$ is in city $i$ and has so far constructed the partial solution $s^p$, the probability of going to city $j$ is given by:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{c_{il} \in N(s^p)} \tau_{il}^\alpha \tau_{il}^\beta} & \text{if } c_{ij} \in N(s^p), \\ 0 & \text{otherwise}, \end{cases} \tag{2-9}$$

where $N(s^p)$ is the set of feasible components: that is the edges $(i,j)$ where $l$ is a city not yet visited by ant $k$. The parameters $\alpha$ and $\beta$ control the relative importance of the pheromone versus the heuristic information $\eta_{ij}$, which is given by:

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{2-10}$$

where $d_{ij}$ is the distance between cities $i$ and $j$.

### 2.4.2 MAX-MIN Ant System (MMAS)

This algorithm [60] is an improvement over the original AS. Its characterizing elements are that only the best ant updates the pheromone trails and that the value of the pheromone is bound. The pheromone update is implemented as follows:

$$\tau_{ij} \leftarrow [(1 - \rho) \cdot \tau_{ij} + \Delta \tau_{ij}^{best}] \frac{\tau_{max}}{\tau_{min}} \tag{2-11}$$

where $\tau_{max}$ and $\tau_{min}$ are respectively the upper and lower bounds imposed on the pheromone, and the operator is
\[ [x]_b^a = \begin{cases} 
  a & \text{if } x > a, \\
  b & \text{if } x < b, \\
  x & \text{otherwise};
\end{cases} \quad (2-12) \]

and \( \Delta t_{ij}^{best} \) is

\[ \Delta t_{ij}^{best} = \begin{cases} 
  \frac{1}{L_{best}} & \text{if } (i,j)\text{belongs to the best tour}, \\
  0 & \text{otherwise},
\end{cases} \quad (2-13) \]

where \( L_{best} \) is the length of the tour of the best ant. This maybe either the best tour found in the current iteration “iteration-best” (\( L_{ib} \)) or the best solution found since the start of the algorithm. “best-so-far” (\( L_{bs} \)) or a combination of both.

### 2.4.3 Ant Colony System (ACS)

The most interesting contribution of ACS [61,62] is the introduction of a “local pheromone update” in addition to the pheromone update performed at the end of the construction process. The local pheromone update is performed by all the ants after each construction step. Each ant applies it only to the last edge traversed:

\[ \tau_{ij} = (1 - \varphi) \cdot \tau_{ij} + \varphi \cdot \tau_0, \quad (2-14) \]

where \( \varphi \in (0,1] \) is the pheromone decay coefficient, and \( \tau_0 \) is the initial value of the pheromone.

The main goal of the local update is to diversify the search performed by subsequent ants during an iteration: by decreasing the pheromone concentration on the traversed edges, ants encourage subsequent ants to choose other edges and, hence, to produce different solutions. This makes it less likely that several ants produce identical solutions during one iteration.
The offline pheromone update, similarly to MMAS, is applied at the end of each iteration by only one ant, which can be either the iteration-best or best-so-far. However, the update formula is slightly different:

$$\tau_{ij} \leftarrow \begin{cases} (1 - \rho) \cdot \tau_{ij} + \rho \cdot \Delta \tau_{ij} & \text{if } (i,j) \text{ belongs to best tour}, \\ \tau_{ij} & \text{otherwise}, \end{cases}$$ (2-15)

As in MMAS,

$$\Delta \tau_{ij} = \frac{1}{L_{\text{Best}}}$$ (2-16)

where $L_{\text{Best}}$ can be either $L_{ib}$ or $L_{bs}$.

Another important difference between ACS and AS is in the decision rule used by the ants during the construction process. In ACS, the so called *pseudorandom proportional* rule is used: the probability for an ant to move from city $i$ to city $j$ depends on a random variable $q$ uniformly distributed over $[0,1]$ and a parameter $q_0$; if $q \leq q_0$, then,

$$j = \arg\max_{c \in \mathbb{N}(s^p)} \left\{ \tau_{ic} \eta_{ic}^\beta \right\}$$ (2-17)

### 2.4.4 Standard Ant Colony Optimization

ACO algorithms use agents called artificial ants which have the properties of real ants but also along with some characteristics that differentiate them from real ants.

- Real ants evaluate the intensity of pheromone during their way from the nest to their food while artificial ants evaluate a solution by determining the intensity of pheromones only when they are travelling back to their nest.
- Real ants might not take the same path on their way to their food sources and on their return trip to their nest but artificial ants always move from the nest to their food sources and return following the same path.

- Real ants lay pheromone when they move away from and back to the nest. However, artificial ants only deposit pheromones on their way back to the nest.

Some capabilities were added to the artificial ants.

- Memory is used by artificial ants to save the path that they have taken while constructing their solutions. Then, they drop pheromones with an intensity that depends on the quality of the solutions. The better the solutions, the more pheromones.

- The transition policy of artificial ants not only depends on the pheromone trail but also on specific heuristic information. Real ants choose their path with respect to the pheromones only.

- Pheromone evaporation is added for artificial ants to prevent the colony from trapping in a suboptimal solution whereas in real ant colonies, pheromone evaporation is too slow to be a significant part of their search mechanism.

```
Ant colony optimization metaheuristic

Set parameters, initialize pheromone trails

while termination conditions not met do
    ConstructAntSolutions
    ApplyLocalSearch {optional}
    UpdatePheromones
end while
```

Figure 2-4: Basic principles of Ant Colony Optimization (ACO).

Basically, the ACO metaheuristic is composed of 3 main phases: the solution construction phase, update pheromone phase and daemon actions.
1. The solution construction phase is a stochastic transition policy, which is a probability function of the pheromone trail and controls the movement of the ants. After the ants have completed their solutions, they evaluate the quality of the solutions, which will be used in the pheromone update phase.

2. The pheromone update phase biases the search to the area that have high-quality solutions. In this phase, the pheromone trails are adjusted based on the previous paths the ants traveled. The update phase consists of decreasing and increasing the pheromone intensity of the trails. Decreasing the pheromone intensity can be done through pheromone evaporation while increasing pheromone intensity is implemented by adding pheromone on the paths used in the solutions of the ants in the previous iteration. The amount of pheromone that is deposited depends on the quality of the solutions for the ants’ respective paths. The paths that are used in many solutions or have better solutions receive more pheromones, therefore, the intensity of pheromones will be biased towards the better solutions.

3. The daemon actions phase is an optional phase in which there are extra actions to the original solutions or a centralized action is implemented. For this dissertation, daemon actions are not used.

2.4.5 Ant Colony Optimization for Continuous Variable Domains

Even though ACO was originally introduced for combinatorial optimization [63], it has been developed to solve continuous optimization problems. These problems are characterized by the fact that decision variables have continuous domains, in contrast to discrete problems. In 2008, K. Socha and M. Dorigo [64] proposed a new approach to ACO for continuous domains (ACOR). The fundamental idea was the shift from using a discrete probability distribution to using a continuous one, that is a probability density function (PDF).
The ACO metaheuristic method finds approximate solutions to an optimization problem by iterating the following two steps:

1) Candidate solutions are constructed in a probability way using a probability distribution over the search space.

2) The candidate solutions are used to modify the probability distribution in a way that is deemed to bias future sampling toward high quality solutions.

Fig 2-5: (a) Discrete probability distribution of a finite set \( \{c_{i1}, \ldots, c_{i10}\} \in N(s^p) \) of available components. (b) Continuous probability density function \( P_c(x|s^p) \) with possible range \( x \in [x_{\text{min}}, x_{\text{max}}] \).

In ACO for combinatorial problems, the pheromone values are associated with a finite set of discrete values related to the decisions that the ants make. This allows ACO to represent the pheromone values in the form of a pheromone table. This is not possible in the continuous case, as the number of possible values is not finite. Hence, \( ACO_R \) rather uses a “solution archive” as a way of describing the pheromone distribution over the search space. The solution archive contains a number of complete solutions to the problem. While a pheromone model in combinatorial optimization can be seen as an implicit memory of the search history, a solution archive is an explicit memory.

The basic flow of the \( ACO_R \) algorithm is as follows. As a first step, the solution archive is initialized. The weight is defined to be a value of a Gaussian function with argument \( l, \text{mean} = 1 \), and standard deviation \( q_k \), where \( q \) is a parameter of the algorithm. When \( q \) is small, the best ranked solutions are strongly preferred, and when it is large, the probability becomes more uniform.
\[ p_t = \frac{\omega}{\sum_{r=1}^{\infty} \omega_r} \] (2-18)

\[ \omega = \frac{1}{q^2 \sqrt{2\pi}} e^{-\frac{(t-1)^2}{2q^2k^2}} \] (2-19)

Then, at each iteration, a number of solutions is probabilistically constructed by the ants. These solutions may be improved by any improvement mechanism (e.g., local search or gradient techniques). Finally, the solution archive is updated with the generated solutions. In the following, we outline the components of ACO\(_R\) in more detail.

Fig 2-6: The structure of the solution achieves. The solutions in the archive are sorted according to their quality.

### 2.4.6 Modified Continuous Ant Colony Optimization using Prospect Theory

Another variation of ACO\(_R\) that is used in this dissertation comes from a Ph.D. thesis, Cognitive Ant Colony Optimization: A New Framework in Swarm Intelligence by Indra Chandra Joseph Riadi [65]. In this thesis, Riadi proposed a new framework of ACO based on prospect theory (PT) that modifies the state transition during the solution construction phase. Instead of directly using a random generator it uses PT to choose the next position. PT looks at two parts of this state transition: the editing and the evaluation phase. In the editing phase, the objective function is framed as losses or gains relative to the reference...
point, while in the evaluation phase the decision maker will choose options that are influenced by their subjected value and perceptual likelihood.

In this framework, during the solution construction process, a solution is constructed by searching its variables one by one until all variables are found. To do this, a solution is first chosen not only from its weight alone but also from its objective function. The reference point is the average of the value of the objective function of all solutions in the solution achieve.

\[ \text{Reference point} = \text{mean}(f(S_1, S_2, \cdots, S_k)) \]  

(2-20)

Next, the value function is derived from the objective function \( f(S_1, S_2, \cdots, S_k) \). If the objective function of a particular solution is less than the reference point, then the solution is in the loss region; conversely, if the objective function is larger than the reference point, then the solution is in the gain region.

The value function is formulated as:

\[ v(x_1) = \begin{cases} 
\alpha x_1^\alpha & \text{for } x_1 \geq 0 \\
-\lambda(-x_1)^\beta & \text{for } x_1 < 0 
\end{cases} \]  

(2-21)

where \( \alpha > 0, \beta > 0, \lambda > 0, \)

\[ x_i = f(S_i) - \text{Reference point}, \alpha = \beta = 0.88 \text{ and } \lambda = 2.77; l = 1, \ldots k \]

Next we calculate the probability weighting function using the prospect theory. In the normal ACO, the solution is chosen based on the probability function shown in the equation above, however, in this framework, the weighting probability function is changed as follows:

\[ \pi(p_i) = \frac{\rho_i^\gamma}{(\rho_i^\gamma + (1-\rho_i)^\gamma)^\gamma}, \text{where } \gamma = 0.68 \]  

(2-22)

Finally, the prospect of each choice is calculated

\[ V_l = v(x)\pi(p_l) \]  

(2-23)

The best solution is the candidate solution with the largest prospect. This solution will be used to calculate the amount of pheromone.
Input: \( k, m, n, q, \xi \) and termination criterion

Output: The best solution found

// Initialize and evaluate \( k \) solutions
\( T = \text{Sort}(S_1 \ldots S_k) \)

While Termination criterion is not satisfied

Do

// Generate \( m \) new solutions

For \( i = 1 \) to \( m \)

Do

// Construct solution

For \( j = 1 \) to \( n \)

Do

// Calculate the reference point
\( \text{Reference point} = \text{mean}(f(S_1 \ldots S_k)) \)

// Calculate the value function of PT
\( (objval_1^i \ldots objval_k^i) = \text{valuefunction}(\text{Reference point}, (s_1^i \ldots s_k^i)) \)

// Calculate the probability weighting function of PT
\( (wp_1^i \ldots wp_k^i) = \text{wp-function}(p(w_1^i \ldots w_k^i)) \)

// Calculate the prospect of PT
\( (pros_1^i \ldots pros_k^i) = \text{pros-function}((objval_1^i \ldots objval_k^i), (wp_1^i \ldots wp_k^i)) \)

\( \text{maxprospect} = \max(pros_1^i \ldots pros_k^i) \)

Select Gaussian \( g_1^i \) according to maxprospect

Sample Gaussian \( g_1^i \) with parameter \( \mu_1^i, \sigma_1^i \)

End

Store and evaluate newly generated solution

End

// Sort solutions and select the best \( k \)
\( T = \text{Best}(\text{Sort}(S_1 \ldots S_k), k) \)

End

Figure 2-7: Pseudo-code of ACOR-PT algorithm.
2.5 Firefly Algorithm (FA)

2.5.1 Introduction to the Firefly Algorithm

The Firefly Algorithm (FA) mimics the behavior of fireflies and their flashing light, which is produced by a process of bioluminescence [27]. The fundamental function of such flashes is to attract mating partners.

For FA, the light can be formulated in such a way that it is associated with the objective function to be optimized. That is the intensity of light $l$ goes on decreasing as the distance $r$ increase in terms of $l \propto 1/r^2$. Additionally, air absorbs the light, which becomes weaker with increasing distance. These two factors combined make most fireflies visible at a limited distance.

The differences between real fireflies and artificial fireflies in this algorithm are

- Artificial fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex, while real fireflies are attracted to their opposite sex.

- The attractiveness of another artificial firefly is proportional to its brightness; thus, for any two flashing fireflies, the less bright one will move towards the brighter one. Also, as attractiveness is proportional to brightness, both factors decrease as the fireflies’ distance increases and if there is no other firefly brighter than a particular firefly, it will move randomly.

- The brightness of an artificial firefly is affected by the characteristic of the objective function. For example, in a maximization problem, the brightness is proportional to the value of the objective function.
2.5.2 Standard Firefly Algorithm (SFA)

Two important issues in the SFA are the variation of light intensity and the formulation of attractiveness. In general, the attractiveness of a firefly is determined by its brightness, which is associated with the objective function. In the simplest case of maximizing the objective function \( f(x) \), the brightness \( I \) of a firefly at a location \( x \) can be chosen as \( I(x) \propto f(x) \). However, attractiveness \( \beta \) should be chosen in relation to other fireflies. Thus, it will vary with the distance \( r_{ij} \) between firefly \( i \) and firefly \( j \). In addition, light intensity decreases with the distance from its source, and light is also absorbed in the media, so we should allow attractiveness to vary with the absorption coefficient \( \gamma \). In the simplest form, the light intensity varies according to the inverse square law.

\[
I(r) = \frac{I_s}{r^2}
\]  
(2-24)

where \( I_s \) is the intensity at the light source. For a given medium with a fixed light absorption coefficient \( \gamma \), the light intensity \( I \) varies with the distance \( r \) by

\[
I = I_0 e^{-\gamma r}
\]  
(2-25)

where \( I_0 \) is the original light intensity. In order to avoid the singularity at \( r = 0 \) in the expression \( I_s/r^2 \), the combined effect of both the inverse square law and absorption can be approximated as the following Gaussian form

\[
I(r) = I_0 e^{-\gamma r^2}
\]  
(2-26)

As a firefly’s attractiveness is proportional to the light intensity seen by other fireflies, we can define the attractiveness \( \beta \) of a firefly by

\[
\beta = \beta_0 e^{-\gamma r^2}
\]  
(2-27)
where \( \beta_0 \) is the attractiveness at \( r = 0 \). As it is often faster to calculate \( 1/(1 + r^2) \) than an exponential function, the above function, if necessary, can conveniently be approximated as

\[
\beta = \frac{\beta_0}{1 + \gamma r^2}
\] (2-28)

If we define a characteristic distance \( \Gamma = 1/\sqrt{\gamma} \), the attractiveness changes from \( \beta_0 \) to \( \beta_0 e^{-1} \) and to \( \beta_0/2 \) for equations above. In practical terms, the attractiveness function \( \beta(r) \) can be any monotonically decreasing functions such as the following generalized form.

\[
\beta(r) = \beta_0 e^{-\gamma r^m}
\] (2-29)

where \( m \geq 1 \)

For a fixed \( \gamma \), the characteristic length becomes

\[
\Gamma = \gamma^{-1/m} \to 1, \quad m \to \infty
\] (2-30)

Conversely, for a given length scale \( \Gamma \) in an optimization problem, the parameter \( \gamma \) can be used as a typical initial value. That is

\[
\gamma = \frac{1}{\Gamma^m}
\] (2-31)

The distance between any two fireflies \( i \) and \( j \) at \( x_i \) and \( x_j \) is Cartesian distance

\[
r_{ij} = \|x_i - x_j\| = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2}
\] (2-32)

where \( x_{i,k} \) is the \( k \)th component of the spatial coordinate \( x_i \) of \( i \)th firefly. For example, in a 2-dimension case, we have

\[
r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\] (2-33)
The movement by which a firefly \( i \) is attracted to another more attractive (brighter) firefly \( j \) is determined by

\[
x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2} (x_j - x_i) + \alpha \epsilon_i
\]

(2-34)

where the second term is due to the attraction and third term is the randomness with \( \alpha \) being the randomization parameter and \( \epsilon_i \) is a vector of random numbers drawn from a Gaussian distribution or uniform distribution.

\[\textbf{Objective function} f(x), x = (x_1, \ldots, x_d)^T\]

\[\text{Generate initial population of fireflies } x_i \ (i = 1, 2, \ldots, n)\]

\[\text{Light intensity } I_i \text{ at } x_i \text{ is determined by } f(x_i)\]

\[\text{Define light absorption coefficient } \gamma\]

\[\text{While } (t < \text{MaxGeneration})\]

\[\text{For } i = 1 : n \text{ all } n \text{ fireflies}\]

\[\text{For } j = 1 : n \text{ all } n \text{ fireflies}\]

\[\text{If } (I_i < I_j), \text{ Move firefly } i \text{ towards } j;\]

\[\text{End}\]

\[\text{Vary attractiveness with distance } r \text{ via } e^{-\gamma r}\]

\[\text{Evaluate new solutions and update light intensity}\]

\[\text{End}\]

\[\text{End}\]

\[\text{Rank the fireflies and find the current global best}\]

\[\text{End}\]

Figure 2-8: Pseudo-code of the Firefly Algorithm (FA).

The distance \( r \) defined above is not limited to the Euclidean distance, and we can define another distance \( r \) in the \( n^{th} \) dimension depending on the type of problem. The typical scale \( \Gamma \) should be associated with the scale concerned in our optimization. If \( \Gamma \) is the typical scale for a given optimization
problem, for a very large number of fireflies $n \gg k$ where $k$ is the number of local optima, then the initial locations of these $n$ fireflies should distribute relatively uniformly over the entire search space.

As the iterations proceed, the fireflies would converge into all the local optima including the global optima. By comparing the best solutions among all the optima, the global optima can easily be found. The convergences speed of FA can be adjusted by varying the parameters $\gamma$ and $\alpha$.

### 2.5.3 Adaptive Firefly Algorithm (AdaFA)

Population-based algorithms such as FA use the information explored to a certain generation within a population. However, their efficiency depends on the characteristics of the population’s diversity. The greater the diversity of the population the greater the search power of the population-based algorithm. Therefore, parameters that are appropriate at the beginning of optimization can become inappropriate for later generations. The idea of an adapting control parameter during optimization arose [66,67] and has overgrown into the self-adaptation of control parameters, where the parameters are encoded into genotypes of individuals and undergo operations of the variation operators [68].

FA involves three control parameters: the randomization parameter $\alpha$, the attractiveness parameter $\beta$, and the light absorption coefficient $\gamma$. All these parameters are encoded into real-valued vectors in the following form:

$$x_i^{(t)} = (x_{i0}^{(t)}, ... x_{in}^{(t)}; \alpha^{(t)}, \sigma_0^{(t)}; \beta^{(t)}, \sigma_1^{(t)}; \gamma^{(t)}, \sigma_2^{(t)}), \quad \text{for } i = 1, ..., NP$$

(2-35)

where the first part of vector $x_i^{(t)} = (x_{i0}^{(t)}, ... x_{in}^{(t)})$ represents a position of the $i$th firefly, similar to SFA, and parameters $\alpha^{(t)}, \beta^{(t)}$, and $\gamma^{(t)}$ are the current values of the control parameters, while $\sigma_0^{(t)}, \sigma_1^{(t)}$ and $\sigma_2^{(t)}$ are their standard deviations.
The self-adaptive parameters undergo an operation of uncorrelated mutation with three-step sizes [69]. This mutation is described by the following equations:

\[
\sigma_i^{(t+1)} = \sigma_i^{(t)} \cdot e^{\left(\tau'N(0,1)+\tau N_i(0,1)\right)}
\] (2-36)

\[
\alpha^{(t+1)} = \alpha_i^{(t)} + \sigma_0^{(t)} \cdot N(0,1)
\] (2-37)

\[
\beta^{(t+1)} = \beta_i^{(t)} + \sigma_1^{(t)} \cdot N(0,1)
\] (2-38)

\[
\gamma^{(t+1)} = \gamma_i^{(t)} + \sigma_2^{(t)} \cdot N(0,1)
\] (2-39)

where \(i = 1, \ldots, 3\) and \(\tau' \propto 1/\sqrt{2 \cdot \sqrt{n}}\) denotes the so-called learning rate. Here, the rule that prevents the mutation strength \(\sigma_i^{(t)}\) from falling under a certain minimum value \(\varepsilon_0\) is applied

\[
\sigma_i^{(t)} < \varepsilon_0 \Rightarrow \sigma_i^{(t)} < \varepsilon_0,
\] (2-40)

for \(i = 1, \ldots, 3\)
2.6 Other Nature-Inspired Optimization Methods

Beyond the four nature-inspired optimization methods that have been discussed, many other methods exist, such as the Bat Algorithm (BA) [70], Differential Evolution (DE) [71], and Invasive Weed Optimization (IWO) [72]. BA is based on the behavior of microbats with varying pulse rates of emission and loudness. Each virtual bat flies randomly with a velocity $v_i$ and position $x_i$ and has a varying frequency or wavelength and loudness. As it searches for its prey, it changes frequency, loudness and pulse emission rate.

DE [73] is a novel heuristic approach for optimizing nonlinear and non-differentiable continuous space functions. The population of DE consists of vectors randomly initialized from a uniform distribution with each vector representing a solution. The population evolves in each generation with the use of mutation, crossover and selection.

IWO is a novel numerical optimization technique inspired by colonizing weeds, which are very robust and adaptive to changes in the environment [72]. Starting with a finite number of seeds being spread over a search area, as every seed grows to a flowering plant, they produce seeds depending on their fitness. The new seeds are randomly dispersed over the search area continuously until a maximum number of plants is reached. At this point, only the plants with the highest fitness can survive and produce new seeds, whereas the others are eliminated.
Chapter 3

Impedance Matching Problem

Figure 3-1: A load connected to a transmission line through a matching network.

By definition [74], impedance matching is the connection of additional impedance elements to existing ones in order to achieve a specific effect, such as maximum power transfer from the output of one circuit to the input of another, or to reduce reflection in the transmission line or balance a circuit.

Depending on the application, matching maybe required at a single frequency or over a band of frequencies such that the bandwidth of the matching network is involved in the objective function. If the load impedance varies over a given range, a matching network can be adjusted or tuned as necessary. In general, matching networks are constructed with reactive components only so that no loss is added to the overall network.
### 3.1 Broadband Matching Problem

A crucial task in transmitters, amplifiers, receivers, antennas and other RF applications is the design of an impedance matching equalizer network as shown in Figure 3-2.

![Figure 3-2: Two-ports broadband matching network.](image)

The goal is to transfer power from the source to the load by transforming a complex load impedance \( Z_L = R_L + jX_L \) to match a resistive or complex source impedance \( Z_S = R_S + jX_S \) over a wide frequency band.

These impedances are usually measured at a finite number of radio frequencies. A sinusoidal source voltage \( E \) at any particular frequency is applied to lossless equalizer input port 1 through \( Z_S \), which can provide the maximum-available source power \( P_{as} \) to the load \( Z_L \) when input impedance \( Z_{in} = R_{in} + jX_{in} = Z_S^* = R_S - jX_S \) (conjugate of \( Z_S \)). Otherwise, there is some power mismatch, which is the per unit-power reflected by the equalizer. Power mismatch is also expressed as return loss

\[
R_L = -20 \log(MM^2)
\]  

(3-1)

The goal is to find an equalizer network that minimizes the mismatch, thus maximizing the transducer power gain. Two tools crucial to broadband matching were described in 1939 and have been
relevant ever since. Darlington’s Theorem says that an impedance function of an arbitrary assemblage of reactive and resistive elements can be represented by a reactive (lossless L and C) network terminated in a 1-ohm resistance [75]. If this is applied to Figure 3-2, for $Z_l = 1 + j0$ there is always an LC network that can produce any impedance function $Z_{in}(p)$ versus a complex frequency $p = \sigma + j\omega$ that is rational positive real. A positive real impedance function $Z_{in}$ has $R_{in} > 0$ when $\sigma > 0$ and $X_1 = 0$ when $\omega = 0$. Positive real impedance functions occur as the ratio of specific polynomials in the complex frequency variable $p$. Darlington’s Theorem is evidently false if the 1-ohm termination is replaced by any other impedance, which poses the compatible impedance problem [76].

The Smith chart, initially conceived in 1939 [77] for transmission-line analysis, is a transformation of all impedances in the right-half Argand plane (RHP) into a unit circle (Smith, 1939). That bilinear transformation was originally just as in equation (5) with the Smith chart center $Z_l = 1 + j0$ ohms, but is equally applicable to (2) or (3) where the chart center corresponds to $Z_s^*$ or $Z_l^*$, respectively. That curved distance metric on a Smith chart (geodesic) also can be expressed by the voltage standing-wave ratio (VSWR), which is a scaled version of the hyperbolic distance or mismatch as opposed to the original transmission-line voltage interpretation [78].

Originally, the Smith chart was designed to display lines of constant resistances and reactances. It can also be used to design impedance matching at a single frequency by moving the load point towards the center, the source point, along constant reactance circles. Later, however, many engineers started to apply the Smith chart to simple broadband impedance matching network designs. For each frequency, an impedance point is marked on the Smith chart to form a curve.
3.2 Impedance Matching Network Design

A general purpose of designing an impedance matching network (IMN) is to make the signal’s energy transfer as much as possible from source to load within a broad frequency range. This goal is realized by means of inserting a lossless two ports matching network between the source and the load, as shown in Figure 3-3.

![Matching Network Diagram]

Figure 3-3: The matching network is inserted between the source and load.

The performance of the IMN in Figure 3-3 can be evaluated by a parameter called transducer power gain (TPG), which is defined by Equation (1):

\[ TPG = \frac{4R_q R_l}{(R_q + R_l)^2 + (X_q + X_l)^2} \]  
\[ \Gamma = \frac{(Z_{in} - Z_s)}{(Z_{in} + Z_s)} \]

where \( Z_l = R_l + jX_l \) is the load impedance, \( Z_q = R_q + jX_q \) is the impedance seen from \( Z_l \) to the matching network, and \( Z_s \) is the source impedance.

Through a simple calculation, the value of TPG is between 0 and 1. A common purpose of the matching network is to make the TPG as big as possible within a specified frequency range. There are two main types of broadband impedance matching methods: the analytical method and the CAD method.
3.2.1 Analytical Methods

The analytical method originated from Bode’s theoretical research on impedance matching [79]. Bode studied the matching problem of RC parallel loads and obtained the gain-bandwidth limitation of the problem. Fano [80] developed Bode’s theory and solved the matching problem for any kind of load in a more general way. This work culminated in a principle known as Bode-Fano Limit.

The Bode-Fano Limit

There exists a general limit on the bandwidth over which an arbitrarily good impedance match can be obtained in the case of a complex load impedance. This general limit is related to the ratio of reactance to resistance, and to the bandwidth over which we wish to match the load. Bode and Fano derived, for lumped circuits, a fundamental limitation that is expressed for a parallel RC load impedance as

$$\int_0^\infty \ln \left| \frac{1}{|\Gamma(\omega)|} \right| d\omega \leq \frac{\pi}{RC}$$  (3-4)

Since $\ln(1) = 0$, there is no contribution to this integral over frequencies for which $|\Gamma| = 1$, so we can see that we want to have the maximum mismatch out of the band of interest. If we assume this condition, the integral is limited to the bandwidth of interest, and we can get an idea of how well we can match an arbitrary complex impedance over that bandwidth.

Consider the idealized situation shown in the figure 3.4, where the reflection coefficient is unity outside the band of interest and is $\Gamma_m$ in the frequency range $\Delta\omega$. 
From simplification, the integral becomes

\[ \Delta \omega \ln \frac{1}{\Gamma_m} \leq \frac{\pi}{RC} \]  

(3-5)

From \( \ln(x) = -\ln(1/x) \), we can solve for \( \Gamma_m \)

\[ \Gamma_m \geq e^{-1/2\Delta fRC} \]  

(3-6)

Therefore, this solves for the maximum bandwidth for which a given RC product can be matched to within a given reflection coefficient or SWR. In general, the Bode-Fano limits can be written as

\[ \frac{BW}{\omega_0 \Gamma_{avg}} \leq \frac{\pi G}{B} \]  

(3-7)

\[ \frac{BW}{\omega_0 \Gamma_{avg}} \leq \frac{\pi R}{X} \]  

(3-8)

where \( G \) is the load conductance, \( B \) is the load susceptance, and \( X \) is the load reactance. This can be written in terms of the load quality factor \( Q \) as follows:

\[ \frac{BW}{\omega_0 \Gamma_{avg}} \leq \frac{\pi}{Q} \]  

(3-9)

It can be seen from the above expressions that the more reactive energy that is stored in a load, the narrower the bandwidth of a match. Also, the higher the \( Q \) is, the narrower the bandwidth of the match for the same average in-band reflection coefficient.
Figure 3-5: Impedance matching networks with reactive loads. a) Parallel RC load. b) Parallel RL load. c) Series RC load. d) Series RL load.

Youla reconsidered this problem based on complex normalization theory and if the analytical form of a load is known, a matching network can be designed for any impedance matching problem using Fano and Youla’s theory. However, the analytical method is very complicated, and besides, most of the time the analytical form of the load is unknown. These shortcoming limit the Youla theory’s practice in engineering.
3.2.2 CAD Methods

In order to overcome the limitations of analytical theory, Carlin has proposed a new method called Real Frequency Technique (RFT), which removes the major difficulties from which the analytic theory suffers [81]. Later on Carlin, Yarman, Fettweis and Pandel also developed some alternative real frequency algorithms [82,83,84]. These are line-segment technique, direct computational technique, parametric representation of Brune functions, and the scattering approach. These methods use the experimental real frequency data for the generator and the load, and it is not necessary to assume either the analytic form of the transfer function or the equalizer topology. It has been also shown for various single matching problems, that compared to a design obtained with analytic theory, RFT results in better performance with simpler structures [85,86,87]. Because of these advantages, RFT has become the most feasible approach to solve broadband matching problems. However, by the invention of efficient and accurate data modeling tools, matching network design by analytic methods remains an unanswered problem for the researchers.

Another different CAD method is the Recursive Stochastic Equalization (RSE) method [88]. Based on an initially imposed topology, TPG is obtained as the objective function that has a function relationship with the frequency and impedance of the matching network. A stochastic Gauss-Newton algorithm followed by a limited random.

A relatively simple method commonly applied in RF analog circuit design is based on Smith charts. It assumes that the bandwidth has an inverse relationship with the quality factor $Q$. Most likely, it is true because the impedance changes within a small range and has only one peak within its working frequency range, but this is not the case when the method is applied to some loads such as a broadband piezoelectric transducer (BPT) [89]. The impedance of a BPT usually has multiple peak values and changes greatly, in which case the inverse relationship between bandwidth and quality factor is no longer true.

From the above discussion, it can be concluded that IMNs synthesized by current typical current methods still might have room for improvement.
3.2.3 Network Topologies

Design issues of matching networks

1) Complexity: The simplest design that satisfies the required specification is generally the most preferable (cheaper, more reliable, and less lossy).

2) Bandwidth: Normally, it is desirable to match a load over a band of frequencies. Increased bandwidth usually comes with increased complexity, e.g. using multistage matching.

3) Implementation: It is necessary to choose the right type of matching networks, lumped elements, tuning stubs, and transmission lines.

4) Adjustability: This is required for applications where a variable load impedance occurs.

There are three basic matching networks used in matching network designs: L, T and Pi networks that are subsets of longer many element ladder networks.

**L network**

![L network topology](image)

Figure 3-6: L network topology.
**T network**

![T network diagram](image)

Figure 3-7: T network topology.

**Pi Network**

![Pi network diagram](image)

Figure 3-8: Pi network topology.
3.3 Lossy Impedance Matching

The broadband matching problem is defined as the transfer of power from source to load by transforming a complex load impedance to match a resistive or complex source impedance over a wide frequency band. The matching networks are usually designed to work at a single frequency but in some cases, e.g., antenna impedance matching, we need an antenna that works over a wide frequency band. Conjugate matching can be used in single frequency matching network design but is not physically possible over a finite frequency band. By adding loss, however, more bandwidth may be acquired at a cost in the transducer power gain of the matching networks. We refer to Figure 3-9 to address the broadband matching problem in further detail. The voltage source $V_g$ is sinusoidal at a particular frequency, with maximum power that can be delivered to the network $P_{as}$.

![Figure 3-9: Typical circuit configuration with a lossy two-ports network between the source and the load.](image)

The source impedance is $Z_s = R_s + jX_s$, and the load impedance is $Z_l = R_l + jX_l$. Power absorbed by the load is $P_l$. The power mismatch $M_1$ is the per-unit reflectance by the two-port network looking from the source side, and $M_2$ is that looking from the load side.

$$M_1 = \frac{|Z_1-Z_s|}{Z_1+Z_s}$$  \hspace{1cm} (3-14)
\[ M_2 = \frac{|Z_2 - Z_1|}{|Z_2 + Z_1|} \quad (3-15) \]

The transducer power gain (TPG) is obtained from
\[ TPG = \frac{P_L}{P_{as}} = 1 - M_1^2 M_2^2 - NetworkLoss \quad (3-16) \]

When the two-ports network in Figure 3.9 is lossless, the two goals, to "maximize the power transferred to the load" and "minimize the power reflected from the load," are equivalent.

This can also be expressed in the following way. There is no loss in the two-port,
\[ NetworkLoss = 0, \text{ thus,} \]
\[ TPG = 1 - M_1^2 M_2^2. \quad (3-17) \]

When we try to maximize the transducer power gain (TPG), we are automatically minimizing the product \( M_1^2 M_2^2 \). When both ports of the network are matched, the real power absorbed by the load is the same as the power entering the lossless passive network
\[ Z_1 = Z_5^*, Z_1 = Z_2^*, M_1 = M_2 = 0, \quad (3-18) \]

Note that this is also the difference between \( P_{as} \) and the reflected power. Following the above reasoning, it is clear that when there is network loss in the two-port, "maximizing the power transferred to the load" does not necessarily mean "minimizing the power reflected from the load." According to equation (4), the following scenario could happen: assuming both ports are matched, then
\[ M_1 = M_2 = 0 \quad (3-19) \]
\[ TPG = 1 - NetworkLoss \quad (3-20) \]

We can see that even when there is no mismatch whatsoever in the circuit, the transducer gain could still be very low if the network is lossy. Apparently, when mismatch exists in the circuit, either \( M_1 \neq 0 \) or \( M_2 \neq 0 \) or more practically, neither of them is zero, and the transducer gain can only go down from the value of 1 - Network Loss. When we are "maximizing the power transferred to the load," we want to maximize \( G_T \); when we are "minimizing the power reflected from the load," we want to minimize \( M_1 \) and \( M_2 \). These are now two separate goals because of the network loss. In other words, we
need to keep the loss because we want a wide frequency band; meanwhile, we also need to control the loss so that it does not absorb an appreciable amount of power.

To avoid the domination of one objective over the other, it is important to keep the range of the two goals close. That is why we need to normalize the objectives before putting them into one equation as the final objective function to be minimized. In the lossy impedance matching problem, we also need to take the range of the two objectives into consideration. Now let us revisit the goals:

a) Maximize power transferred to the antenna;

b) Minimize power reflected from the antenna.

As we discussed earlier, these two objectives, which are the same thing in lossless impedance matching problems, are now separate and conflicting because of the lossy component inserted in the matching network.

Assuming that the power available from the source is 1, it can be divided into three parts: power reflected from the lossy network, power absorbed by the lossy component in the network, and power transferred to the antenna. The transducer power gain, $G_T$ can be defined as the ratio of the power delivered to the load to the maximum power deliverable to the network from the generator. Pozar [90] gives an equation for the transducer power gain:

$$ G_T = \frac{P_{load}}{P_{max}} $$

Now we consider the first goal: minimize reflection. We have the reflection coefficient $\Gamma$ also directly related to the reflection. The absolute value of $\Gamma$ is $(0,1)$, so we write the first objective as

$$ J_1 = |\Gamma| $$

(3-21)

The range for the transducer gain is $(0,1)$. Since the optimization algorithm performs minimization, here we write the second objective as

$$ J_2 = 1 - G_T $$

(3-22)

Now that we have two normalized objectives, we can write a final objective function according to the equation

$$ \min \lambda J_1 + (1 - \lambda)J_2 \text{ where } 0 \leq \lambda \leq 1 $$

(3-33)
The weight can be adjusted to determine which is the main objective we want to minimize. In extreme cases like $\lambda = 1$, the objective becomes only the first goal: to maximize the reflection; when $\lambda = 0$, we only minimize the transducer power gain.
Chapter 4

Comparison of Nature-Inspired Optimization Methods for Impedance Matching Problem

4.1 Design and Simulation

The test problem is an 80-meter high-frequency dipole antenna. The procedure begins by importing the real antenna impedance measurement over 3.5-3.85 MHz from a text file into MATLAB. Then we choose the network topologies, which consist of many options. The Pi and T networks are more complicated than L networks as they have 3 lumped elements rather than 2. Therefore, we use the impedance matching of a Pi3 network.

Figure 4-1: Pi3 network.

The objective function is to minimize the maximum VSWR over 3.5-3.85 MHz. Many optimization methods were used. First, the global optimization toolbox from MATLAB included 3 methods: pattern search, simulated annealing and GA. Each algorithm was used in various settings with numbers of loops of 50, 100, 500, 1000, 5000 and restarting each algorithm’s starting points. We then recorded each algorithm’s best result maximum VSWR, minimum VSWR, lumped elements value at the optimum point and their computation time.
Next, we ran the simulation without a loop of the same problem using the nature-inspired optimization algorithms. MATLAB codes for GA, PSO, ACO, and FA were written appropriately for this particular problem. Various parameters were varied in order to compare for their best results, such as number of populations and generations for GA, number of particles/ants/fireflies and iteration for Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Firefly Algorithm (FA), respectively.

Each algorithm’s best result consists of the maximum and minimum VSWR over frequency 3.5 to 3.85 MHz, its lumped elements value at converged points, and its computation time. Lastly, graphs of VSWR over the frequencies 3.4-4 MHz were plotted and the results from all simulations were compared.
4.2 Global Optimization Toolbox in MATLAB

For pattern search and simulated annealing optimization, we used a MATLAB simulation code based on a thesis, Impedance Matching Optimization Based on MATLAB by Kaiming Li [91], which used the global optimization toolbox from MATLAB.

Table 4-1: VSWR, time, and lumped elements value result of loop optimizer by pattern search and simulated annealing from the MATLAB global optimization toolbox.

<table>
<thead>
<tr>
<th>Solver</th>
<th>Pattern Search</th>
<th>Simulate Annealing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loops</td>
<td>100</td>
<td>5000</td>
</tr>
<tr>
<td></td>
<td>100</td>
<td>5000</td>
</tr>
<tr>
<td>C1/pF</td>
<td>752.82</td>
<td>922.2649</td>
</tr>
<tr>
<td></td>
<td>986.9156</td>
<td>4.32e03</td>
</tr>
<tr>
<td>C2pF</td>
<td>1.52e03</td>
<td>1.56e03</td>
</tr>
<tr>
<td></td>
<td>1.75e03</td>
<td>8.18e03</td>
</tr>
<tr>
<td>L/nH</td>
<td>2.82e03</td>
<td>2.60e03</td>
</tr>
<tr>
<td></td>
<td>2.45e03</td>
<td>655.9938</td>
</tr>
<tr>
<td>Time/min</td>
<td>11s</td>
<td>9:08</td>
</tr>
<tr>
<td></td>
<td>2:34</td>
<td>2:40:19</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.9276</td>
<td>1.9049</td>
</tr>
<tr>
<td></td>
<td>1.8664</td>
<td>1.3477</td>
</tr>
</tbody>
</table>
Figure 4-2: VSWR over 3.4-4 MHz with 100 loops using pattern search from the MATLAB global optimization toolbox.

Figure 4-3: VSWR over 3.4-4 MHz with 5000 loops using pattern search from the MATLAB global optimization toolbox.
Figure 4-4: VSWR over 3.4-4 MHz with 100 loops using simulate annealing from the MATLAB global optimization toolbox.

Figure 4-5: VSWR over 3.4-4 MHz with 5000 loops using simulate annealing from the MATLAB global optimization toolbox.
We used GA from the MATLAB Global Optimization Toolbox in two configurations population = 20 and population = 200 with a varying number of loops of 100, 500, 1000 and 5000.

Table 4-2: VSWR, time, and lumped elements value result of Loop optimizer by GA with population = 20 from the MATLAB global optimization toolbox.

<table>
<thead>
<tr>
<th>Loops</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1/pF</td>
<td>376.98</td>
<td>1.51e3</td>
<td>979.96</td>
<td>1.94e3</td>
</tr>
<tr>
<td>C2pF</td>
<td>1.18e3</td>
<td>3.55e3</td>
<td>1.94e3</td>
<td>4.01e3</td>
</tr>
<tr>
<td>L/nH</td>
<td>3.54e3</td>
<td>1.61e3</td>
<td>2.38e3</td>
<td>1.34e3</td>
</tr>
<tr>
<td>Time/min</td>
<td>29s</td>
<td>2:26</td>
<td>4:46</td>
<td>15:30</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>2.0386</td>
<td>1.776</td>
<td>1.8587</td>
<td>1.6261</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.3149</td>
<td>1.2085</td>
<td>1.4218</td>
<td>1.2954</td>
</tr>
</tbody>
</table>
Figure 4-6: VSWR over 3.4-4 MHz with 100 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20.

Figure 4-7: VSWR over 3.4-4 MHz with 500 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20.
Figure 4-8: VSWR over 3.4-4 MHz with 1000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20.

Figure 4-9: VSWR over 3.4-4 MHz with 5000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 20.
Table 4-3: VSWR, time, and lumped elements value results of a loop optimizer by GA with population = 200 from the MATLAB global optimization toolbox.

<table>
<thead>
<tr>
<th>Loops</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1/pF</td>
<td>777.8127</td>
<td>2.01e03</td>
<td>2.92e03</td>
<td>3.67e03</td>
</tr>
<tr>
<td>C2/pF</td>
<td>1.60e03</td>
<td>3.23e03</td>
<td>5.38e03</td>
<td>7.05e03</td>
</tr>
<tr>
<td>L/nH</td>
<td>2.75e03</td>
<td>1.42e03</td>
<td>956.4791</td>
<td>757.2178</td>
</tr>
<tr>
<td>Time/(Hour:Min:Sec)</td>
<td>00:03:06</td>
<td>00:18:53</td>
<td>00:35:14</td>
<td>3:13:12</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.9199</td>
<td>1.5616</td>
<td>1.3877</td>
<td>1.3428</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.4353</td>
<td>1.5195</td>
<td>1.2909</td>
<td>1.1655</td>
</tr>
</tbody>
</table>

Figure 4-10: VSWR over 3.4-4 MHz with 100 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200.
Figure 4-11: VSWR over 3.4-4 MHz with 500 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200.

Figure 4-12: VSWR over 3.4-4 MHz with 1000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200.
Figure 4-13: VSWR over 3.4-4 MHz with 5000 loops using GA from the MATLAB global optimization toolbox with a parameter population = 200.
4.3 MATLAB-based Code Simulation

4.3.1 Genetic Algorithm (GA)

In this case, we used GA that is not from MATLAB Global Optimization Toolbox. Instead, we used a MATLAB-based simulation code written from the standard GA. We used two configurations: the first configuration consisted of a population = 70 and generation = 150, and the second configuration consisted of a population = 150 and generation = 150. We ran the simulation 1000 times for each configuration. Here we show 3 examples of the simulation results.

Table 4-4: VSWR, time, and lumped elements value result by GA.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>population = 70, generation = 150</th>
<th>population = 150, generation = 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1/pF</td>
<td>3.52E+03  4.08E+03  3.63E+03</td>
<td>3.80E+03  4.09E+03  4.08E+03</td>
</tr>
<tr>
<td>C2/pF</td>
<td>6.74E+03  7.80E+03  6.98E+03</td>
<td>7.29E+03  7.83E+03  7.82E+03</td>
</tr>
<tr>
<td>L/nH</td>
<td>7.90E+02  688.96   764.74</td>
<td>733.83   686.41   688.18</td>
</tr>
<tr>
<td>Time/Second</td>
<td>0.408     0.422    0.337</td>
<td>0.397    0.405    0.407</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.3464    1.3387   1.3435</td>
<td>1.3412   1.3386   1.3386</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.1844    1.1142   1.163</td>
<td>1.1535   1.111    1.1102</td>
</tr>
</tbody>
</table>
Figure 4-14: VSWR over 3.4-4 MHz using GA with a parameter population = 70, generation = 150.

Figure 4-15: VSWR over 3.4-4 MHz using GA with a parameter population = 150, generation = 150.
4.3.2 Particle Swarm Optimization (PSO)

In this case, we use a MATLAB-based simulation code written from the standard PSO. We used two configurations: the first configuration consisted of a population = 70 and iteration = 150 and the second configuration consisted of a population = 150 and iteration = 150. We ran the simulation 1000 times for each configuration. Here we show 3 examples of the simulation result.

Table 4-5: VSWR, time, and lumped elements value results by PSO.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>population = 70, iteration = 150</th>
<th>population = 150, iteration = 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1/pF</td>
<td>3.91E+03 4.10E+03 4.16E+03</td>
<td>4.14E+03 3.90E+03 3.79E+03</td>
</tr>
<tr>
<td>C2/pF</td>
<td>7.50E+03 7.85E+03 7.95E+03</td>
<td>7.83E+03 7.47E+03 7.26E+03</td>
</tr>
<tr>
<td>L/nH</td>
<td>7.15E+02 685.11 684.35</td>
<td>6.79E+02 715.33 737.03</td>
</tr>
<tr>
<td>Time</td>
<td>1.422s 1.523s 1.523s</td>
<td>2.861s 2.947s 2.914s</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.3412 1.3389 1.3998</td>
<td>1.3524 1.3434 1.3413</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.1844 1.1118 1.048</td>
<td>1.1134 1.134 1.1682</td>
</tr>
</tbody>
</table>
Figure 4-16: VSWR over 3.4-4 MHz using PSO with a parameter population = 70, iteration = 150.

Figure 4-17: VSWR over 3.4-4 MHz using PSO with a parameter population = 150, iteration = 150.
4.3.3 Ant Colony Optimization (ACO)

In this case, we used a MATLAB-based simulation code based on ACOR-PT from a thesis, “Cognitive Ant Colony Optimization: A New Framework in Swarm Intelligence” []. We used two configurations: the first configuration consisted of a population = 70 and iteration = 150 and the second configuration consisted of a population = 150 and iteration = 150. We ran the simulation 1000 times for each configuration. Here we show 3 examples of the simulation results.

Table 4-6: VSWR, time, and lumped elements value results by ACO.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>population = 70, iteration = 150</th>
<th>population = 150, iteration = 150</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C1/pF</td>
<td>C2/pF</td>
</tr>
<tr>
<td>C1/pF</td>
<td>4.02E+03</td>
<td>3.01E+03</td>
</tr>
<tr>
<td>C2/pF</td>
<td>8.51E+03</td>
<td>5.59E+03</td>
</tr>
<tr>
<td>L/nH</td>
<td>6.76E+02</td>
<td>925.86</td>
</tr>
<tr>
<td>Time/Second</td>
<td>1.877</td>
<td>1.816</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.4312</td>
<td>1.3788</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.1211</td>
<td>1.2726</td>
</tr>
</tbody>
</table>
Figure 4-18: VSWR over 3.4-4 MHz using ACO with a parameter population = 70, iteration = 150.

Figure 4-19: VSWR over 3.4-4 MHz using ACO with a parameter population = 150, iteration = 150.
4.3.4 Firefly Algorithm (FA)

In this case, we used MATLAB-based simulation code from the original book for the standard FA. We used two configurations: the first configuration consisted of a population = 70 and iteration = 150 and the second configuration consisted of a population = 150 and iteration = 150. We ran the simulation 1000 times for each configuration. Here we show 3 examples of the simulation results.

Table 4-7: VSWR, time and lumped elements value results by FA.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>population = 70, iteration = 150</th>
<th>population = 150, iteration = 150</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1/pF</td>
<td>3.43E+03</td>
<td>4.01E+03</td>
</tr>
<tr>
<td>C2/pF</td>
<td>6.55E+03</td>
<td>7.67E+03</td>
</tr>
<tr>
<td>L/nH</td>
<td>8.10E+02</td>
<td>700.12</td>
</tr>
<tr>
<td>Time/Second</td>
<td>1.072</td>
<td>1.118</td>
</tr>
<tr>
<td>Max VSWR</td>
<td>1.3497</td>
<td>1.339</td>
</tr>
<tr>
<td>Min VSWR</td>
<td>1.1993</td>
<td>1.1252</td>
</tr>
</tbody>
</table>
Figure 4-20: VSWR over 3.4-4 MHz using FA with a parameter population = 70, iteration = 150.

Figure 4-21: VSWR over 3.4-4 MHz using FA with a parameter population = 150, iteration = 150.
4.4 Results and Analysis

Table 4-8: VSWR and accuracy results from simulation of GA, PSO, ACO, and FA, run 1000 times each.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>GA</th>
<th>PSO</th>
<th>ACO</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Lowest max VSWR</td>
<td>1.3386</td>
<td>1.3386</td>
<td>1.3386</td>
<td>1.3386</td>
</tr>
<tr>
<td>Average max VSWR</td>
<td>1.3419</td>
<td>1.3398</td>
<td>1.3552</td>
<td>1.3467</td>
</tr>
<tr>
<td>99 percent accuracy</td>
<td>981</td>
<td>995</td>
<td>512</td>
<td>807</td>
</tr>
<tr>
<td>98 percent accuracy</td>
<td>997</td>
<td>1000</td>
<td>618</td>
<td>860</td>
</tr>
<tr>
<td>97 percent accuracy</td>
<td>1000</td>
<td>1000</td>
<td>984</td>
<td>990</td>
</tr>
</tbody>
</table>

“99 percent accuracy” is a result that is within one percent range of optimal value

Comparing the results from the MATLAB global optimizer for pattern search, simulated annealing and GA from Figure 4.2 to Figure 4.13, and Table 4-1, 4-2, and 4-3, GA has a better result than pattern search and simulated annealing. Moreover, GA with a configuration population = 200 has a better result than in a configuration with population = 20. As we increase the loops, we get a better result, especially in simulated annealing and in, GA in the first and second settings which achieve maximum VSWR values from 1.87 to 1.35, 2.0 to 1.62, and 1.91 to 1.34, respectively. However, overall the result from the MATLAB global optimizer is not good in terms of Max VSWR, in that only GA with 5000 loops achieves the near optimum value at 1.34. In this case, it took 3 hours and 13 minutes’ computation time.
From Figure 4.14 to Figure 4.21, we can see that GA, PSO, ACO, and FA have similar results in terms of lumped element value and the maximum VSWR graph over 3.4-4 MHz in both configurations (population = 70, iteration = 150 and population = 150, iteration = 150). From Table 4-4 to Table 4-7, we see in the first configuration that GA has the lowest average computation time at 0.396 second, followed by FA, PSO and ACO at 1.098, 1.489 and 1.83 seconds, respectively. In the second configuration, GA also has the lowest computation time at 0.401 second, followed by FA at 1.102 second, while PSO and ACO computation times double their first configuration computation time at 2.907 and 2.963 seconds, respectively.

In terms of accuracy, from Table 4-8 for the first configuration, GA has the lowest average maximum VSWR at 1.3419, followed by FA and PSO, which have average maximums of 1.3432 and 1.3552, respectively. However, ACO has a relatively high average maximum VSWR at 1.5302. In terms of consistency for the first configuration, GA and FA have 98.1% and 90.5% of their results with maximum VSWR within a one-percent range of the optimal value, but PSO and ACO have only 51.2% and 52.1% of results within a one-percent range of the optimal value, respectively. Moreover, in the second configuration, all four algorithms have acceptably consistent results at 99.5%, 80.7% 83.3% and 90.7% for GA, PSO, ACO and FA, respectively.

Compared to the MATLAB global optimizer results which have a wide range of result, in the second configuration, the four nature-inspired algorithms have 90 percent of their results within a three percent range of the optimal value. Furthermore, all four nature-inspired algorithms performed much better than the MATLAB global optimization toolbox in terms of computation time (2895 to 29242 times faster). PSO and ACO have consistent problems in the first configuration, but if we increase the population from 70 to 150, they can perform well. In conclusion, the impedance matching designer can obtain a near optimum impedance matching network when using a nature-inspired algorithm with the right configuration.
Chapter 5

A New Modified Firefly Algorithm for Optimization Problems

5.1 Standard Firefly Algorithm (SFA)

The SFA, that was introduced by Xin-She Yang [92] is based on the characteristic of fireflies, insects that produce short and rhythmic flashes by a process of bioluminescence. The function of the flashing light is to attract partners (communication) or potential prey and to serve as a protective warning toward predators. Thus, the intensity of light is the factor that causes other fireflies to move toward a given firefly. For simplicity, the flashing characteristic can be described by way of the following three rules:

1) Fireflies are attracted toward each other regardless of gender.

2) The attractiveness of the fireflies is correlative with their brightness thus a less attractive firefly will move forward to a more attractive firefly.

3) The brightness of fireflies depends on the objective function

In SFA, two important variables are the light intensity and attractiveness. A firefly is attracted to another firefly that has a brighter flash than itself. This attractiveness depends upon the light intensity and the distance between fireflies.
Pseudo code of the SFA is given below:

Maximize \( f(\mathbf{x}), \mathbf{X} \in \mathbb{R}^d \)

\( f(X) \) is an objective function, \( X_i = (x_1, \ldots, x_d)^T \)

Generate initial population of fireflies \( X_i, i = 1, 2, \ldots, N \)

Light intensity \( I_i \) at \( x_i \) is determined by \( f(X_i) \)

Define \( \beta_0, \gamma, \) and \( \alpha \)

while \( t < \text{MaxGeneration} \)

Rank the fireflies in increasing order of \( I_i \)

for \( i = 1 : N - 1 \)

for \( j = i + 1 : N \)

Move firefly \( i \) towards \( j \) according to equation (4)

Keep the new solution that has the highest light intensity end for \( j \)

update \( X_i \) and \( f(X_i) \)

end for \( i \)

Rank the fireflies in increasing order of \( I_i \)

end while

Postprocess results and visualization
Light intensity is inversely proportional to the particular distance $r$ from the light source. Thus, the light decreases as the distance increases as follows.

$$I(r) = I_0 e^{-\gamma r^2} \quad (5-1)$$

$I(r)$ = light intensity at a distance $r$ from the light source,

$I_0$ = original light intensity,

$\gamma$ = the light absorption coefficient

The attractiveness of firefly $j$ to firefly $i$ is defined as

$$\beta_{ij} = \beta_0 e^{-\gamma r_{ij}^2} \quad (5-2)$$

where $\beta_0$ is the attractiveness at $r_{ij} = 0$

The distance between firefly $i$ and firefly $j$ is defined as

$$r_{ij} = |X_i - X_j| = \sqrt{\sum_{k=1}^{d}(X_{i,k} - X_{j,k})^2} \quad (5-3)$$

Firefly $i$ is attracted towards the more attractive firefly $j$, and the movement is defined as

$$X_i^{t+1} = X_i^t + \beta_0 e^{-\gamma r_{ij}^2} (X_j^t - X_i^t) + \alpha \epsilon_i \quad (5-4)$$

In equation (5-4), $\gamma$ and $\alpha$ are very important parameters. If $\gamma$ approaches zero ($\gamma \to 0$), the attractiveness is constant, $\beta_{ij} = \beta_0$, which means that every firefly can be seen in any position. If the $\gamma$ is nearing infinity or is too large ($\gamma \to \infty$), the attractiveness is zero and the firefly movements become random. The implementation of SFA can be done through these two asymptotic behaviors. While the second term is for randomization, as $\alpha$ is the randomize parameter, the $\epsilon_i$ can be specified as $(\text{ran} - 1/2)$ when ran is the random number generated with a value between 0 and 1.
The SFA is widely used to solve many problems, such as solving the economic emissions load dispatch problem [93], solving multilevel image thresholding selection [94], finding optimal test sequence generation [95], solving the travelling salesman problem [96], vector quantization for image compression [97] and solving object tracking [98].

Despite that it is widely used for solving problems, the SFA has some shortcomings in terms of trapping into several local optima when solving complex problems [99,100]. There are some disadvantages of using only a single method, because it will be overly restrictive for high dimensional and nonlinear problems. Thus, some modifications and hybridizations are suggested to overcome the shortcoming of a single method [101,102,103].
5.2 Modified Firefly Algorithm (MFA)

The MFA improves the SFA in three points:

1) The computational domain is scaled into a unit hyper cube $u^d$, where $0 \leq u \leq 1$, $d$ is the dimension of the searching domain.

2) The attractiveness of firefly $i$ toward firefly $j$, $\beta_{ij}$, is modified as

$$
\beta_{ij} = \beta_0 l_j e^{-\gamma r^2} \tag{5-5}
$$

$$
r = r_{ij}/\sqrt{d} \tag{5-6}
$$

$$
l_j = (f(x_j) - f_{min})/f_{max} - f_{min}) \tag{5-7}
$$

$f_{max}$ is the maximum of $f(X_j)$, $j = 1, \ldots, N$

$f_{min}$ is the minimum of $f(X_j)$, $j = 1, \ldots, N$

In the simulation, the typical value of $\beta_0$ is one. When firefly $i$ is attracted towards the more attractive firefly $j$, the movement is defined as

$$
X_i^{t+1} = X_i^t + \beta_{ij} (X_j^t - X_i^t) + \alpha \epsilon_i \tag{5-8}
$$

$$
X_i^{t+1} = X_i^t + \beta_0 l_j e^{-\gamma r^2} (X_j^t - X_i^t) + \alpha \epsilon_i \tag{5-9}
$$

3) In the SFA, the attractiveness of fireflies will be a constant when $\gamma$ approaches 0, that is, when a given firefly can be seen by all the others. The movement of fireflies is in the so-called the exploitation mode. In this case, FA is the same as a classical PSO. If $\gamma$ approaches $\infty$, the attractiveness will be equal to 0. All the fireflies cannot take their bearings to move except in random flight. The movement of the firefly is in the so-called the exploration mode. In this case, FA becomes a pure random search algorithm.
We want the exploration to increase, but also to have the exploration decrease as the number of iterations increases. Thus, \( \gamma \) and \( \alpha \) should be gradually reduced. Both parameters can be formulated as

\[
\gamma = \gamma_0 (\theta_1)^t \\
\alpha = \alpha_0 (\theta_2)^t
\]  \hspace{1cm} (5-10)

Where \( 0 < \theta_1, \theta_2 < 1 \), \( t \) is the iteration. \( \gamma_0 \), and \( \alpha_0 \) are the initial value of \( \gamma \), and \( \alpha \), respectively.

In practice, we want to have \( \gamma \) and \( \alpha \) to be very small values at the end of iteration.

Let,

\[
\gamma = \gamma_0 (\theta_1)^t = \varepsilon_1 \\
\alpha = \alpha_0 (\theta_2)^t = \varepsilon_2
\]  \hspace{1cm} (5-12)

at \( t = Max\_Iteration \), thus,

\[
\theta_1 = e^{(\log(\varepsilon_1/\gamma_0)/Max\_Iteration)} \\
\theta_2 = e^{(\log(\varepsilon_2/\alpha_0)/Max\_Iteration)}
\]  \hspace{1cm} (5-14)

For example,

If \( \varepsilon_1 = 0.001 \), \( Max\_iteration = 50 \), \( \gamma_0 = 25 \), we will have \( \theta_1 = 0.8167 \)

If \( \varepsilon_2 = 0.001 \), \( \alpha_0 = 0.03 \), we will have \( \theta_2 = 0.8922 \)
Pseudo code of the MFA for the maximization problem is given below:

\[ f(X) \text{ is an objective function, } X = (x_1, ..., x_d)^T \]

\[ \text{Generate initial population of fireflies } x_i, i = 1, 2, ..., N \]

\[ \text{Light intensity } l_i \text{ at } x_i \text{ is determined by } f(X_i) \]

\[ \text{Define } \beta_0, \gamma_0, \text{ and } \alpha_0 \]

\[ \text{while } (t < \text{Max\_iteration}) \]

\[ \text{Rank the fireflies in increasing order of } l_i \]

\[ \text{for } i = 1 : N - 1 \]

\[ \text{for } j = i + 1 : N \]

\[ \text{Move firefly } i \text{ towards } j \text{ according to equation (5)} \]

\[ \text{Keep the new solution that has the highest light intensity end for } j \]

\[ \text{update } X_i \text{ and } f(X_i) \]

\[ \text{end for } i \]

\[ \text{Rank the fireflies in increasing order of } l_i \]

\[ \text{end while} \]

\[ \text{Postprocess results and visualization} \]
5.3 Parameter Selection Guidelines

For simplicity for parameter tuning, one can use $\beta_0 = 1$; therefore, the two parameters to be tuned are $\gamma_0 > 0$ and $0 < \alpha_0 < 1$. One can see that $\gamma$ controls the scaling and it indicates how fast amplitude can fall, while $\alpha$ controls the randomness.

Consider $e^{-\gamma r^2}$, we define a characteristic length, $\Gamma$, as the value of $r$ such that $e^{-\gamma r^2} = e^{-1}$. This means that $\gamma = 1/\Gamma^2$, $0 < \Gamma < 1$.

Then, if we choose $\Gamma$, $\gamma$ can be determined by

$$\gamma = 1/\Gamma^2$$

(5-16)

In this dissertation, the program is run with $20 < \gamma_0 < 200$, $0.01 < \alpha_0 < 0.8$. Thus, it is expected that only $\gamma_0$ and $\alpha_0$ require fine-tuning, while other values (i.e., $\theta_1$, $\theta_2$, and $\beta_0$) do not need to be tuned carefully.
5.4 Optimization of Test Functions

Define the formulas that will be used to compute the number of iterations \(Max_it\) and number of the firefly population \(N\) as follows:

\[
Max_it = 20 \cdot \text{dim} + 10 \tag{5-17}
\]

\[
N = 5 \cdot \text{dim} + 40 \tag{5-18}
\]

where \(\text{dim}\) is the number of dimension of objective function.

5.4.1 Two-dimensional Functions

The performance of the new MFA was tested for its performance in finding maximum solutions with six two-dimension benchmark functions [104]: The Ackley, Rastrigin, sphere, Michalewicz, Yang and four-peaks functions.

The Ackley Function

The formula of Ackley Function [105] is shown below

\[
f(x, y) = -20\exp(-0.2\sqrt{0.5x^2 + y^2}) - \exp(0.5(\cos(2\pi x) + \cos(2\pi y))) + e + 20 \tag{5-19}
\]

The search space is restricted to \(-5 \leq x, y \leq 5\). The global minimum of this function is equal to zero, attained at \((0,0)\).

The plot of \(-f(x, y)\) is shown in Figure 5-1. The locations of fireflies after 10 and 50 iterations are shown in Figures 5-2 and 5-3, respectively. A comparison of the SFA and MFA cost functions is shown in Figure 5-4.
Figure 5-1: The Ackley function.

Figure 5-2: Locations of fireflies after 10 iterations.
Figure 5-3: Final locations of fireflies after 50 iterations, converging into (0,0).

Figure 5-4: Comparison of the performance of SFA and MFA for minimization of the Ackley function with $\gamma_0 = 25$, and $\alpha_0 = 0.02$. 
**Sphere Function**

The formula of sphere function is shown below

\[ f(x, y) = x^2 + y^2 \]  \hspace{1cm} (5-20)

where \(-\infty \leq x \leq \infty, -\infty \leq y \leq \infty\)

The global minimum of this function is equal to zero, attained at (0,0). The plot of \(-f(x, y)\) is shown in Figure 5-5. The locations of fireflies after 10 and 50 iterations are shown in Figures 5-6 and 5-7, respectively. A comparison of the SFA and MFA cost functions is shown in Figure 5-8.

![Figure 5-5: The sphere function.](image)
Figure 5-6: Locations of fireflies after 10 iterations.

Figure 5-7: Final locations of fireflies after 50 iterations, converging into (0,0).
Figure 5-8: Comparison of the performance of SFA and MFA with $\gamma_0 = 150$, and $\alpha_0 = 0.3$.

**Rastrigin Function**

The formula of Rstrigin Function [106] is shown below

$$f(x, y) = 20 + (x^2 - 10 \cdot \cos(2 \cdot \pi x)) + (y^2 - 10 \cos(2 \cdot \pi y)) \tag{5-21}$$

The global minimum of this function is equal to zero, attained at (0,0). The plot of $-f(x, y)$ is shown in Figure 5-9. The locations of fireflies after 10 and 50 iterations are shown in Figures 5-10 and 5-11, respectively. A comparison of the SFA and MFA cost functions is shown in Figure 5-12.

Figure 5-9: The Rastrigin function.
Figure 5-10: Locations of fireflies after 10 iterations.

Figure 5-11: Final locations of fireflies after 50 iterations, converging into (0,0).
Figure 5-12: Comparison of the performance of SFA and MFA with $\gamma_0 = 30$, $\alpha_0 = 0.1$, and $Max_i t = 500$.

**Michalewicz Function**

The formula of Michalewicz Function [107] is shown below

$$f(x, y) = \sin(x) \cdot (\sin\left(\frac{x^2}{\pi}\right)^{2m} + \sin(y) \cdot (\sin\left(\frac{y^2}{\pi}\right)^{2m})$$ (5-22)

The search space is restricted to $-5 \leq x, y \leq 5$. The global minimum of this function is equal to -1.80, attained at (2.20, 1.57). The plot of $-f(x, y)$ is shown in Figure 5-13. The locations of fireflies after 10 and 50 iterations are shown in Figure 5-14 and 5-15, respectively. A comparison of the SFA and MFA cost functions is shown in Figure 5-16.
Figure 5-13: The Michalewicz function.

Figure 5-14: Locations of fireflies after 10 iterations.
Figure 5-15: Final locations of fireflies after 50 iterations, converging into (2.20,1.57).

Figure 5-16: Comparison of the performance of SFA and MFA with $\gamma_0 = 20$, and $\alpha_0 = 0.02$. 
Yang’s Forest Function

The formula of Yang’s Forest Function [108] is shown below

\[ f(x, y) = (|x| + |y|) \cdot e^{(-\sin(x^2) - \sin(y^2))} \]  

(5-23)

The global maximum of this function is equal to zero, attained at (0,0). The plot of \(-f(x, y)\) is shown in Figure 5-17. The locations of fireflies after 10 and 50 iterations are shown in Figures 5-18 and 5-19, respectively. A comparison of the SFA and MFA cost functions is shown in Figure 5-20.

Figure 5-17: The Yang’s forest function.
Figure 5-18: Locations of fireflies after 10 iterations.

Figure 5-19: Final locations of fireflies after 50 iterations, converging into (0,0).
Figure 5-20: Comparison of the performance of SFA and MFA with \( \gamma_0 = 10, \alpha_0 = 0.02 \).

Four-Peaks Function

The formula of Four-Peaks Function is shown below

\[
 f(x, y) = (|x| + |y|) \cdot e^{-x^2-y^2} \tag{5-24}
\]

The global maximum of this function is equal to 0.606, attained at (-0.5, -0.5), (0.5, -0.5), (-0.5, 0.5), and (0.5, 0.5). This function needs a fine-tuning of \( \rho \) in order to locate all four peaks. The best result was obtained at \( \rho = 0.545 \). The plot of \( f(x, y) \) is shown in Figure 5-21. The locations of fireflies after 10 and 50 iterations are shown in Figure 5-22 and 5-23, respectively. A comparison of the SFA and MFA cost function is shown in Figure 5-24.
Figure 5-21: The four-peaks function.

Figure 5-22: Locations of fireflies after 10 iterations.
Figure 5-23: Final locations of fireflies after 50 iterations, converging into (-0.5, -0.5), (0.5, -0.5), (-0.5, 0.5), and (0.5, 0.5).

Figure 5-24: Comparison of performance of the SFA and MFA for maximization of the four-peaks function with $\gamma_0 = 10$ and $\alpha_0 = 0.02$. 

5.4.2 Higher-dimensional Functions

The performance of the new MFA was tested for its performance in finding maximum solutions with higher-dimensional benchmark functions: Styblinski-Tang, and Helical-Valley functions have three-dimension, Wood and Powell functions have four-dimension, and Rosenbrock function has five-dimension.

Styblinski-Tang Function

The formula of Styblinski-Tang Function is shown below

\[
f(X) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}
\]  

(5-25)

Search space is \(-5 \leq x_i \leq 5, i = 1,2,3\).

The global minimum of this function is equal to -117.4979, attained at (-2.903, -2.903, -2.903). A comparison of the SFA and MFA cost functions is shown in Figure 5-25.
Figure 5-25: Comparison of performance of the SFA and MFA for minimization of Styblinski-Tang function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 55$, $\text{Max_it} = 70$.

# function calls = 107,800, and execution time= 22.797 sec. for MFA, and 19.9011 sec. for SFA.

**Helical-Valley Function**

The formula of Helical-Valley Function is shown below

$$f(X) = 100 \left[ x_3 - 10\theta(x_1, x_2) \right]^2 + \left( \sqrt{x_1^2 + x_2^2} - 1 \right)^2 + x_3^2$$  \hspace{1cm} (5-26)

Search space is $-5 \leq x_i \leq 5$, $i = 1, 2, 3$.

The global minimum of this function is equal to zero, attained at $(1,0,0)$. A comparison of the SFA and MFA cost functions is shown in Figure 5-26.
Figure 5.26: Comparison of performance of the SFA and MFA for minimization of Helical-Valley function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, N = 55, Max_it = 70.

# function calls = 107,800, and execution time = 18.368 sec. for MFA, and 18.131 sec. for SFA.

**Wood Function**

The formula of Wood Function is shown below

$$f(X) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$$  \hspace{1cm} (5-27)

Search space is $-5 \leq x_i \leq 5, i = 1,2,3,4$.

The global minimum of this function is equal to zero, attained at (1,1,1,1). A comparison of the SFA and MFA cost functions is shown in Figure 5.27.
Figure 5-27: Comparison of performance of the SFA and MFA for minimization of Wood function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 60$, $\text{Max}_\text{it} = 90$.

# function calls = 164,700, and execution time= 21.288 sec. for MFA, and 22,7013 sec. for SFA.

**Powell Function**

The formula of Powell Function is shown below

$$f(X) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$  \hspace{1cm} (5-28)

Search space is $-5 \leq x_i \leq 5, i = 1,2,3,4$.

The global minimum of this function is equal to zero, attained at (0,0,0,0). A comparison of the SFA and MFA cost functions is shown in Figure 5-28.
Figure 5-28: Comparison of performance of the SFA and MFA for minimization of Powell function with $\gamma_0 = 100$, $\alpha_0 = 0.5$, $N = 60$, $\text{Max}_\text{it} = 90$.

# function calls = 164,700, and execution time= 20.975 sec. for MFA, and 21.297 sec. for SFA.

**Rosenbrock Function**

The formula of Rosenbrock Function is shown below

$$f(X) = \sum_{i=1}^{4}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

(5-29)

Search space is $-30 \leq x_i \leq 30$, $i = 1, 2, 3, 4, 5$.

The global minimum of this function is equal to zero, attained at (1,1,1,1). A comparison of the SFA and MFA cost functions is shown in Figure 5-28.
Figure 5-29: Comparison of performance of the SFA and MFA for minimization of Rosenbrock function with $\gamma_0 = 100$, $\alpha_0 = 0.1$, $N = 500$, Max_it = 20.

# function calls = 2,505,000, and execution time= 275.644 sec. for MFA, and 257.277 sec. for SFA.
5.5 Test Results and Analysis

A new MFA that is proposed here was tested on six standard test functions with two-dimension: The Ackley, sphere, Rastrigin function, Michalewicz function, Yang’s forest function, and four-peaks functions.

1) Ackley function: From Figures 5-2 to 5-3, we can see that after 10 iterations, half of the MFA agents found the optimal value, and after the final iteration, all MFA agents were at the optimal point. Figure 5.4 shows that after 7 iterations, MFA found the best cost at the optimal point (0,0), whereas SFA required 10 iterations to get the same result.

2) Sphere function: From Figures 5-6 to 5-7, we can see that after 10 iterations, half of MFA agents found the optimal value, and after the final iteration, all MFA agents are at the optimal point. After 15 iterations, MFA got a result lower than the optimal result that SFA found after 50 iterations. Also, after 50 iterations, MFA got a much better result than SFA.

3) Rastrigin function: From Figures 5-10 to 5-11, we can see that after 10 iterations, half of MFA agents found the optimal value and after the final iteration, all MFA agents are at the optimal point. From all 50 iterations, MFA consistently got the cost value lower than SFA, including the final optimal point.

4) Michalewicz function: From Figures 5-14 to 5-15, we can see that after 10 iterations, half of MFA agents found the optimal value, and after the final iteration, all MFA agents are at the optimal point. Both SFA and MFA found the same optimal point; however, MFA required only 2 iterations to get to the optimal point while it took SFA 5 iterations to get the same result.

5) Yang’s forest function: From Figures 5-18 to 5-19, we can see that after 10 iterations, half of MFA agents found the optimal value, and after the final iteration, all MFA agents are at the optimal point. From the first 2 iterations, there are no different in results for SFA and MFA. However, after that, MFA was much faster to reach the optimal point. It took MFA 5 iterations, whereas SFA reached the optimal point at 33th iteration.
6) Four-peaks function: From Figures 5-22 to 5-23, we can see that after 10 iterations, about three quarters of MFA agents are in one of the four optimal points and after the final iteration, all MFA agents are at the optimal point. Both SFA and MFA reach the same optimal cost at about the same time. MFA is only one iteration faster than SFA.

7) Styblinski-Tang function: From Figure 5-25, both MFA and SFA got the same final result, however MFA found after 6 iterations and SFA at 13 iterations. After that, they have the same final optimal cost.

8) Helical-Valley function: From Figure 5-26, both MFA and SFA got similar the best optimal costs from first 33 iterations. From 34th iteration, MFA continue search for the best cost closer to optimal value, however SFA only got the same sub-optimal value.

9) Wood function: From Figure 5-27, both MFA and SFA got similar the best optimal costs from first 28 iterations. From 34th iteration, MFA continue search for the best cost closer to optimal value, however SFA only got the same sub-optimal value.

10) Powell function: From Figure 5-28, both MFA and SFA got similar the best optimal costs from first 23 iterations. From 34th iteration, MFA continue search for the best cost closer to optimal value, however SFA only got the same sub-optimal value.

11) Rosenbrock function: From Figure 5-29, we can see that MFA consistently getting the better result than SFA throughout all iterations. After the last iteration, MFA final result value is closer to the optimal value than SFA.

The performance of the MFA in finding optimal solutions for two-dimension benchmark functions is generally better than SFA in all functions. In terms of the optimal result after 50 iterations, SFA and MFA got the same optimal result in all but two functions (sphere function and Rastrigin function), in which SFA only got the near optimal result. This is because SFA require more than 50 iterations to get closer to the optimal point. From the results of all function results, MFA showed that it has faster convergence speed to the optimal point than SFA. It is worth noting that for the four-peaks
function, the results from both algorithms are only slightly different. This is because the four-peaks function has four optimal points; therefore, it is easier to obtain one of the four optimal points, as all of them are also local optima.

For the optimization of higher-dimension functions, that are more complicated problem, the results show that MFA is clearly better than SFA. Only in Styblinski-Tang function that SFA and MFA found the same optimal result, however MFA is two times faster in getting that result. In three functions (Helical-Valley, Wood, and Powell), SFA only found the sub-optimal point and stop getting better result half way through the simulations. This seem to be because SFA converge into some local optimal points. For Rosenbrock function, which is considered to be a difficult function to find optimal points, MFA consistently found the better result in all iterations. Both MFA and SFA seem to require more time in order to find the optimal point in this five-dimension problem.
Chapter 6

Modified Firefly Algorithm (MFA) Optimization for Impedance Matching Networks

From Chapter 3, it can be concluded that impedance matching networks synthesized using current typical methods still might have room for improvement. This dissertation puts forward to a method of designing broadband impedance matching networks based on MFA. This method can find the components’ values automatically within a set of predefined topology.

The MFA can work directly with experimental or simulated load data, without the need for any analytical description of the load impedance function. Most component of impedance matching networks are lossless, but it is shown that when the number of network elements and their values are limited the incorporation of resistors may be useful in order to achieve the prescribed network performance in terms of VSWR, at the expense of some (prescribed) loss of TPG.

In this dissertation, simulation results include cases of both lossless impedance matching networks and lossy impedance matching networks.
6.1 MFA Optimization for Fano LCR Load

The first MFA example is a standard network problem: Optimally match the “Fano load” using the equalizer circuit shown in Figure 6-1. The canonical Fano load comprises an inductor $L_{fano} = 2.3\ \text{henries}$ in series with the parallel combination of a capacitor $C_{fano} = 1.2\ \text{farads}$ and resistor $R_{fano} = 1\Omega$. The equalizer comprises parallel capacitors $C_1$ and $C_3$ connected by a series inductor $L_2$. Power is delivered to the load from a generator whose internal impedance is $R_g = 2.205\Omega$ (purely resistive).

Figure 6-1: Fano LCR load and equalizer topology.

The equalizer’s component values are to be determined by an optimization algorithm. The objective of the optimization problem is to match the source to the load in the radian frequency range $0 \leq \omega \leq 1\ \text{rad/sec}$ by transferring the maximum possible power to the load at all frequencies. This requirement translates to a “max-min” criterion.
The function to be maximized is the minimum value of $TPG(\omega)$ computed at 21 uniformly spaced frequencies in the interval $0 \leq \omega \leq 1$, where $TPG(\omega)$ is the “Transducer Power Gain (TPG). TPG may be expressed as

$$TPG(\omega) = 1 - |\Gamma_{in}|^2$$ (6-1)

$\Gamma_{in}$ is the reflection coefficient, such that

$$\Gamma_{in} = (Z_{in} - R_g)/(Z_{in} + R_g)$$ (6-2)

where $Z_{in}$ is the input impedance seen by the generator at the equalizer input terminals, and can be obtained from following steps:

$$Z_L = (R_{fano} \cdot \left(\frac{1}{j\omega C_{fano}}\right)/(1 + \frac{1}{j\omega C_{fano}})) + j\omega L_{fano}$$ (6-3)

Since we know that, $L_{fano} = 2.3$ henries, $C_{fano} = 1.2$ farads, and $R_{fano} = 1\Omega$, we have

$$Z_L = (1 \cdot \left(\frac{1}{j\omega 1.2}\right)/(1 + \frac{1}{j\omega 1.2})) + j\omega 2.3$$ (6-4)

$$Z_1 = \left(\frac{1}{j\omega C_2}\right)(Z_L)/(Z_L + \frac{1}{j\omega C_w})$$ (6-5)

$$Z_2 = Z_1 + j\omega L$$ (6-6)

$$Z_{in} = \left(\frac{1}{j\omega C_1}\right)(Z_2)/(\left(\frac{1}{j\omega C_1}\right) + Z_2)$$ (6-7)

Also, internal impedance is $R_g = 2.205\Omega$, thus

$$\Gamma = \frac{Z_{in} - R_g}{Z_{in} + R_g} = \frac{Z_{in} - 2.205}{Z_{in} + 2.205}$$ (6-8)

This problem can be solved using Central Force Optimization (CFO) in [109]. CFO is an optimization evolutionary algorithm that searches a multidimensional decision space for the extrema of an objective function to be maximized. It is based on an analogy to classical particle kinematics in a gravitational field. CFO is inherently deterministic, unlike other widely used metaheuristics.

Let $x(1) = C_1$, $x(2) = C_3$, $x(3) = L_2$ and specify boundaries as follows:

$$0.1 \leq C_1 \leq 10, 0.1 \leq C_3 \leq 10, \text{ and } 0.1 \leq L_2 \leq 10$$
Simulation results using CFO, circuit components are obtained as follows:

\[ C_1 = 0.460 \text{ farads}, \ C_3 = 1.006 \text{ farads}, \text{ and } L_2 = 2.988 \text{ henries} \]

Simulation results using MFA with \( \gamma_0 = 100 \) and \( \alpha_0 = 0.6 \), population = 55, and \( \text{Max\_iteration} = 70 \), an average execution time is 49.84 sec, circuit components are obtained as follows:

\[ C_1 = 0.4178 \text{ farads}, \ C_3 = 0.9779 \text{ farads}, \text{ and } L_2 = 3.0114 \text{ henries} \]

A comparison of TPG results obtained by CFO and MFA are shown in Figure 6-2.

![TPG of Faro Load Equalizer](image)

Figure 6-2: Optimized equalizer TPG with CFO and MFA.

A new objective function \( f \) is specified as in (6.9)

\[ f = \sum_{i=1}^{21} TPG(\omega_i) \]  

(6-9)

With \( \gamma_0 = 100, \alpha_0 = 0.6, \text{ population} = 55, \text{ and } \text{Max\_iteration} = 70 \), an average execution time is 46.77 sec. Optimal circuit component values are obtained as follows:
\[ C_1 = 0.6286 \text{ farads}, \quad C_3 = 1.2117 \text{ farads}, \quad \text{and} \quad L_2 = 2.8028 \text{ henries} \]

A comparison of TPG results obtained by CFO and MFA with a new objective function is shown in Figure 6-3.

Figure 6-3: Optimized TPG results by CFO with a standard objective function and MFA with a new objective function.
6.2 MFA Optimization for Lossless Impedance Matching Network

The Pi3 network as shown in Figure 6-4 will be used for an impedance matching network. The load can be obtained from measurement or a software simulation program. In simulation the antenna load data is in the form of A15.txt and a source resistance is 50 ohms.

With $\gamma_0 = 100$, $\alpha_0 = 0.6$, population = 55, and Max_iteration = 70, an average execution time is 30.72 sec. the search space is as follows:

$$1 \text{nF} \leq C_1 \leq 8 \text{nF}, \ 1 \text{nF} \leq C_2 \leq 8 \text{nF}, \text{and} \ 0.1 \mu\text{H} \leq L_2 \leq 3 \mu\text{H}$$

The optimized component values are obtained as follows:

$$C_1 = 4.09 \text{nF}, \ C_3 = 7.83 \text{nF}, \text{and} \ L = 0.6818 \mu\text{H}$$

VSWR of lossless Pi3 network is shown in Figure 6.5, and VSWR obtained from FA and MFA is shown in Figure 6.6.
Figure 6-5: VSWR of the optimized PI3 network, Min_vswr = 1.1107, and Max_vswr = 1.3386

Figure 6-6: Minimum of Max_VSWR vs. iteration, comparison results between FA and MFA.
6.3 MFA Optimization for Lossy Impedance Matching Network

Figure 6-7 to Figure 6-9 show different topologies of lossy matching network. In this dissertation, the lossy network considered is the Pi3 network shown in Figure 6.10

![Diagram of lossy network topology with one resistor](image)

Figure 6-7: Lossy network topology with one resistor.

![Diagram of lossy network topology with two resistors](image)

Figure 6-8: Lossy network topology with two resistors.

![Diagram of lossy network topology with three resistors](image)

Figure 6-9: Lossy network topology with three resistors.

![Diagram of lossy Pi3 network](image)

Figure 6-10: Lossy Pi3 network.
From Figure 6.10, TPG can be calculated from

\[
TPG = \frac{4 \text{Re}([\text{Re}(Z_L)z_{21}]^2)}{|(Z_S+z_{11})(Z_L+z_{22})-z_{12}z_{21}|^2}
\]  

(6-10)

where \(Z_S\) is the source impedance, \(Z_L\) is the load impedance. \(z_{11}, z_{12}, z_{21}, \) and \(z_{22}\) are values of Z-parameter of the matching network, that can be obtained from the following steps:

\[
Y_{c1} = j\omega C_1, \quad X_{c2} = \frac{1}{j\omega C_2}, \quad Y_{L1} = \frac{1}{j\omega L_1}
\]  

(6-11)

\[
AR = \begin{bmatrix} 1 & R \\ 0 & 1 \end{bmatrix}, \quad AC_1 = \begin{bmatrix} 1 \\ Y_{c1} \end{bmatrix}, \quad AC_2 = \begin{bmatrix} X_{c2} \\ 1 \end{bmatrix}, \quad AL_1 = \begin{bmatrix} 1 \\ Y_{L1} \end{bmatrix}
\]  

(6-12)

\[
\begin{bmatrix} A & B \\ C & D \end{bmatrix} = AR \ast AC_1 \ast AC_2 \ast AL_1
\]  

(6-13)

\[
\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix} = \frac{1}{c} \begin{bmatrix} A & (BC - AD) \\ 1 & -D \end{bmatrix}
\]  

(6-14)

Then, we can find a reflection coefficient (\(\Gamma\)) as following steps

\[
Z_A = (j\omega L \cdot Z_L)/(j\omega L + Z_L)
\]  

(6-15)

\[
Z_B = \frac{1}{j\omega C_2} + Z_A
\]  

(6-16)

\[
Z_C = \left(\frac{1}{j\omega C_1}\right) \cdot Z_B/\left(\frac{1}{j\omega C_1} + Z_B\right)
\]  

(6-17)

\[
Z_{in} = R + Z_C
\]  

(6-18)

\[
\Gamma = \frac{Z_{in} - Z_S}{Z_{in} + Z_S}
\]  

(6-19)

The source resistance, \(Z_S = 50 \ \Omega\), and load data are obtained from A15.txt. The optimization problem is to min \(\lambda f_1 + (1 - \lambda) f_2\) where \(0 \leq \lambda \leq 1\) in the frequency range of \(3.5 \ - 3.85 MHz\).

The MFA program run with \(\gamma_0 = 100, \alpha_0 = 0.6,\) population = 200, and Max ITERATION = 60. Each program run has average execution time of 4.30 minutes. The search space used in the optimization is as follows:

\[
40 \leq C_1 \leq 2000 \text{ pF}, \ 40 \leq C_3 \leq 2000 \text{ pF}, \ 40 \leq L_2 \leq 3000 \text{ nH}, \text{ and } 1 \leq R \leq 1000 \ \Omega
\]

The optimized component values are shown in Table 6.1.
Table 6-1: Optimal component value results of lossy Pi3 network for 11 weight values ($\lambda = 0, 0.1, \ldots, 1$). $\gamma_0 = 25, \alpha_0 = 0.05$, population = 300, and Max_iteration = 20.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$c_1/\rho F$</th>
<th>$c_2/\rho F$</th>
<th>$L/nH$</th>
<th>$R/\Omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>924.4</td>
<td>1670.7</td>
<td>2681.6</td>
<td>29.4</td>
</tr>
<tr>
<td>0.1</td>
<td>1689.0</td>
<td>1786.2</td>
<td>1930.8</td>
<td>13.8</td>
</tr>
<tr>
<td>0.2</td>
<td>1397.4</td>
<td>1908.2</td>
<td>2054.0</td>
<td>23.6</td>
</tr>
<tr>
<td>0.3</td>
<td>1199.7</td>
<td>1777.2</td>
<td>2258.2</td>
<td>8.5</td>
</tr>
<tr>
<td>0.4</td>
<td>1294.5</td>
<td>1572.3</td>
<td>2237.4</td>
<td>28.1</td>
</tr>
<tr>
<td>0.5</td>
<td>886.5</td>
<td>1442.8</td>
<td>2603.1</td>
<td>18.8</td>
</tr>
<tr>
<td>0.6</td>
<td>551.3</td>
<td>1100.9</td>
<td>2824.6</td>
<td>28.0</td>
</tr>
<tr>
<td>0.7</td>
<td>1443.0</td>
<td>1509.4</td>
<td>2009.0</td>
<td>28.7</td>
</tr>
<tr>
<td>0.8</td>
<td>928.8</td>
<td>1285.5</td>
<td>1692.6</td>
<td>40.8</td>
</tr>
<tr>
<td>0.9</td>
<td>1628.5</td>
<td>1503.0</td>
<td>1395.4</td>
<td>42.4</td>
</tr>
<tr>
<td>1.0</td>
<td>1522.0</td>
<td>1581.5</td>
<td>1098.5</td>
<td>45.9</td>
</tr>
</tbody>
</table>

Figure 6-11: VSWR of lossy Pi3 network with weight ($\lambda$) = 0.
Figure 6-12: TPG of lossy Pi3 network with $\text{weight}(\lambda) = 0$.

Figure 6-13: VSWR of lossy Pi3 network with $\text{weight}(\lambda) = 0.1$. 
Figure 6-14: TPG of lossy Pi3 network with $\text{weight}(\lambda) = 0.1$.

Figure 6-15: VSWR of lossy Pi3 network with $\text{weight}(\lambda) = 0.2$. 
Figure 6-16: TPG of lossy Pi3 network with \( \text{weight}(\lambda) = 0.2 \).

Figure 6-17: VSWR of lossy Pi3 network with \( \text{weight}(\lambda) = 0.3 \).
Figure 6-18: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.3.

Figure 6-19: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.4.
Figure 6-20: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.4.

Figure 6-21: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.5.
Figure 6-22: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.5.

Figure 6-23: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.6.
Figure 6-24: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.6.

Figure 6-25: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 0.7.
Figure 6-26: TPG of lossy Pi3 network with weight($\lambda$) = 0.7.

Figure 6-27: VSWR of lossy Pi3 network with weight($\lambda$) = 0.8.
Figure 6-28: TPG of lossy Pi3 network with weight(λ) = 0.8.

Figure 6-29: VSWR of lossy Pi3 network with weight(λ) = 0.9.
Figure 6-30: TPG of lossy Pi3 network with weight(\(\lambda\)) = 0.9.

Figure 6-31: VSWR of lossy Pi3 network with weight(\(\lambda\)) = 1.
Figure 6-32: TPG of lossy Pi3 network with weight(\lambda) = 1.

Figure 6-33: TPG of lossy Pi3 network for \lambda=0, \lambda=0.2, \lambda=0.8, and \lambda=1.
Figure 6-34: VSWR of lossy Pi3 network for $\lambda = 0, \lambda = 0.2, \lambda = 0.8$, and $\lambda = 1$.

Table 6-2: Minimum VSWR, maximum VSWR, minimum TPG, and maximum TPG results of lossy Pi3 network for 11 weight values ($\lambda = 0, 0.1, \ldots, 1$).

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Min_VSWR</th>
<th>Max_VSWR</th>
<th>Min_TPG</th>
<th>Max_TPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.7101</td>
<td>2.6209</td>
<td>0.9543</td>
<td>0.9600</td>
</tr>
<tr>
<td>0.1</td>
<td>1.2693</td>
<td>1.7201</td>
<td>0.9350</td>
<td>0.9446</td>
</tr>
<tr>
<td>0.2</td>
<td>1.1151</td>
<td>1.5337</td>
<td>0.9509</td>
<td>0.9894</td>
</tr>
<tr>
<td>0.3</td>
<td>1.1072</td>
<td>1.4748</td>
<td>0.9389</td>
<td>0.9980</td>
</tr>
<tr>
<td>0.4</td>
<td>1.1382</td>
<td>1.4733</td>
<td>0.9112</td>
<td>0.9916</td>
</tr>
<tr>
<td>0.5</td>
<td>1.1056</td>
<td>1.4696</td>
<td>0.8494</td>
<td>0.9749</td>
</tr>
<tr>
<td>0.6</td>
<td>1.1169</td>
<td>1.3468</td>
<td>0.9362</td>
<td>0.9990</td>
</tr>
<tr>
<td>0.7</td>
<td>1.0585</td>
<td>1.3006</td>
<td>0.7301</td>
<td>0.9256</td>
</tr>
<tr>
<td>0.8</td>
<td>1.1006</td>
<td>1.2607</td>
<td>0.7167</td>
<td>0.9197</td>
</tr>
<tr>
<td>0.9</td>
<td>1.0318</td>
<td>1.0908</td>
<td>0.2916</td>
<td>0.4647</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0023</td>
<td>1.0080</td>
<td>0.0234</td>
<td>0.0429</td>
</tr>
</tbody>
</table>
From Figure 6-2 and 6-3, we can see that Both CFO and MFA have similar results of TPG for the optimized equalizer for the Fano LCR load. The optimal LC circuit components values found by CFO and MFA are very close. These results clearly show that MFA solves the Fano load equalizer problem at least as effectively as CFO.

For the lossy Pi3 impedance matching network, Figure 6-11 to 6-34 and Table 6-2 are indicated that, at almost every frequency point, both VSWR and TPG of the lossy Pi3 network increases, as $\lambda$ decreases. That is, because the objective function of the lossy network problem is a multi-objective function, according to equation $\min(\lambda J_1 + (1 - \lambda)J_2)$. More weight is given to the objective $J_1$ when $\lambda$ increases, and more weight is given to objective $J_2$ when $\lambda$ decreases. In this case, $J_1$ is the VSWR and $J_2$ is the TPG.

Depending on the application, network designers have to decide which objective they should place more emphasis on. Based on that, they can set the $\lambda$ accordingly.
6.4 MFA Optimization for HF Broadband Fan Antenna

The main goal of work in this example is to try to get closer to real world problems, where other factors appear, namely component value restrictions, component losses and tolerances, and explicit VSWR requirement. The fictitious frigate model [110] is shown Figure 6-35, containing a broadband antenna that consists of two symmetric fans rigged from the top of the bow mast to the top of the stern mast. Each arm is composed by three AWG-4 (∅ = 5.189 mm) wires 19 m long, with a spacing of 1 m. The antenna radiation impedance calculated on 225 frequencies in the 2-30 MHz. band in 125 KHz. steps are shown in Figure 6-36.

Figure 6-35: CAD frigate model with the HF fan antenna.
The matching network topology proposed in [110] is shown in Figure 6-37, where $x_2$, $x_4$, $x_3$ and $x_5$ represent inductance, and capacitance, respectively. In [110] it is stated that the capacitors should be in the range from 100 to 2500 pF, while the inductances should lie between 1 to 12 μH.

In this dissertation, the impedance matching problem is solved using the Modified Firefly Algorithm (FMA). The object function to be minimized is

$$F = \lambda |\Gamma| + (1 - \lambda)(1 - TP\Gamma), \quad 0 \leq \lambda \leq 1$$  \hspace{1cm} (6-20)

$\Gamma$ is the reflection coefficient obtained from

$$\Gamma = (Z_{in} - Z_S)/(Z_{in} + Z_S)$$  \hspace{1cm} (6-21)
where $Z_s$ is the source impedance, and $Z_{in}$ is the impedance seen from the source. TPG is the transducer power gain defined by

$$TPG = \frac{4\text{Re}(Z_s)\text{Re}(Z_L)|z_{21}|^2}{|(Z_s + z_{11})(Z_L + z_{22}) - z_{12}z_{21}|^2}$$  \hspace{1cm} (6.22)

where $Z_L = R_L + jX_L$ is the load impedance obtained by approximating its values from Figure 6.36. $Z_s = R_S + jX_S$ is the source impedance. $z_{11}, z_{12}, z_{21},$ and $z_{22}$ are values of $Z$-parameter of the matching network, that can be obtain from the following steps:

$$Y_1 = \frac{1}{j\omega L_1}, Z_1 = \frac{1}{j\omega C_1}, Y_2 = \frac{1}{j\omega L_2}, Y_3 = j\omega C_2$$  \hspace{1cm} (6-23)

$$AR = \begin{bmatrix} 1 & R_1 \\ 0 & 1 \end{bmatrix}; AY_1 = \begin{bmatrix} 1 & 0 \\ Y_1 & 1 \end{bmatrix}; AZ_1 = \begin{bmatrix} 1 & Z_1 \\ 0 & 1 \end{bmatrix};$$

$$AY_2 = \begin{bmatrix} 1 & 0 \\ Y_2 & 1 \end{bmatrix}; AY_3 = \begin{bmatrix} 1 & 0 \\ Y_3 & 1 \end{bmatrix}; A_{trans} = \begin{bmatrix} \frac{1}{n} & 0 \\ 0 & n \end{bmatrix};$$  \hspace{1cm} (6-24)

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = AR * AY_1 * AZ_1 * AY_2 * AY_3 * A_{trans}$$  \hspace{1cm} (6-25)

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{bmatrix} = \frac{1}{c} \begin{bmatrix} A & (BC - AD) \\ 1 & -D \end{bmatrix}$$  \hspace{1cm} (6-26)

All results obtained using MFA with $\gamma_0 = 100$, $\alpha_0 = 0.6$, population = 200, and Max_iteration = 50. Each run of program an average execution time of 5.16 minutes. The search space used in the optimization is as follows.

$$0.1 \leq C \leq 2.5 \text{nF}, \quad 1 \leq L \leq 12 \text{\mu H}, \quad 0.001 \leq n \leq 5, \quad \text{and} \quad 0.001 \leq R_1/50 \leq 5 \text{ \Omega}$$

Optimal component values of the matching network for 11 weight values are shown in Table 6.3.
Table 6-3: Optimal component value results of matching network of HF Broadband Fan Antenna for 11 weight values ($\lambda = 0, 0.1, ..., 1$). $\gamma_0 = 100, \alpha_0 = 0.6$, population = 200, and Max_iteration = 50.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$R_1/50$</th>
<th>$L_1(\mu H)$</th>
<th>$C_1(nF)$</th>
<th>$L_2(\mu H)$</th>
<th>$C_2(nF)$</th>
<th>Transformer turn ratio (n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0010</td>
<td>7.0967</td>
<td>2.1016</td>
<td>4.5241</td>
<td>0.1000</td>
<td>1.4754</td>
</tr>
<tr>
<td>0.1</td>
<td>0.0010</td>
<td>6.1276</td>
<td>2.1057</td>
<td>4.9851</td>
<td>0.1000</td>
<td>1.4851</td>
</tr>
<tr>
<td>0.2</td>
<td>0.0010</td>
<td>5.8633</td>
<td>2.2350</td>
<td>5.8902</td>
<td>0.1000</td>
<td>1.4918</td>
</tr>
<tr>
<td>0.3</td>
<td>0.0010</td>
<td>5.8908</td>
<td>2.0871</td>
<td>5.0420</td>
<td>0.1012</td>
<td>1.4878</td>
</tr>
<tr>
<td>0.4</td>
<td>0.1166</td>
<td>5.6042</td>
<td>2.0750</td>
<td>4.3657</td>
<td>0.1000</td>
<td>1.5326</td>
</tr>
<tr>
<td>0.5</td>
<td>0.3158</td>
<td>5.9142</td>
<td>2.2466</td>
<td>3.7218</td>
<td>0.1000</td>
<td>1.5561</td>
</tr>
<tr>
<td>0.6</td>
<td>0.4068</td>
<td>6.2304</td>
<td>2.1403</td>
<td>2.5549</td>
<td>0.1062</td>
<td>1.6068</td>
</tr>
<tr>
<td>0.7</td>
<td>0.5305</td>
<td>4.4976</td>
<td>2.4966</td>
<td>1.6908</td>
<td>0.1275</td>
<td>1.8262</td>
</tr>
<tr>
<td>0.8</td>
<td>0.7541</td>
<td>1.0000</td>
<td>2.4670</td>
<td>4.2654</td>
<td>0.1783</td>
<td>2.8672</td>
</tr>
<tr>
<td>0.9</td>
<td>0.6805</td>
<td>9.7009</td>
<td>2.4505</td>
<td>1.0000</td>
<td>0.1000</td>
<td>2.0057</td>
</tr>
<tr>
<td>1.0</td>
<td>0.8091</td>
<td>1.0000</td>
<td>2.3320</td>
<td>7.1979</td>
<td>0.4232</td>
<td>3.9499</td>
</tr>
</tbody>
</table>

Table 6-4: Minimum VSWR, maximum VSWR, minimum TPG, and maximum TPG results of HF Broadband Fan Antenna for 11 weight values ($\lambda = 0, 0.1, ..., 1$). with $\gamma_0 = 100, \alpha_0 = 0.6$, population = 200, and Max_iteration = 50.

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>Min_VSWR</th>
<th>Max_VSWR</th>
<th>Min_TPG</th>
<th>Max_TPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.141</td>
<td>3.3069</td>
<td>0.7136</td>
<td>0.9955</td>
</tr>
<tr>
<td>0.1</td>
<td>1.2016</td>
<td>3.965</td>
<td>0.6991</td>
<td>0.9946</td>
</tr>
<tr>
<td>0.2</td>
<td>1.278</td>
<td>4.1489</td>
<td>0.675</td>
<td>0.9869</td>
</tr>
<tr>
<td>0.3</td>
<td>1.1998</td>
<td>4.1374</td>
<td>0.6399</td>
<td>0.9714</td>
</tr>
<tr>
<td>0.4</td>
<td>1.0364</td>
<td>4.0262</td>
<td>0.5520</td>
<td>0.9870</td>
</tr>
<tr>
<td>0.5</td>
<td>1.014</td>
<td>3.6251</td>
<td>0.4685</td>
<td>0.9881</td>
</tr>
<tr>
<td>0.6</td>
<td>1.0311</td>
<td>2.9976</td>
<td>0.3736</td>
<td>0.9803</td>
</tr>
<tr>
<td>0.7</td>
<td>1.1165</td>
<td>2.6112</td>
<td>0.3186</td>
<td>0.9681</td>
</tr>
<tr>
<td>0.8</td>
<td>1.0353</td>
<td>2.1105</td>
<td>0.1129</td>
<td>0.8750</td>
</tr>
<tr>
<td>0.9</td>
<td>1.0555</td>
<td>1.9484</td>
<td>0.0961</td>
<td>0.9557</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0268</td>
<td>1.8964</td>
<td>0.0349</td>
<td>0.9044</td>
</tr>
</tbody>
</table>
Figure 6-38: VSWR of HF Broadband Fan Antenna for $\lambda=0$, $\lambda=0.2$, $\lambda=0.8$, and $\lambda=1$.

Figure 6-39: TPG of HF Broadband Fan Antenna for $\lambda=0$, $\lambda=0.2$, $\lambda=0.8$, and $\lambda=1$. 
For HF Broadband Fan Antenna matching network, Figure 6-39 to 6-40 and Table 6-4 are indicated that

1) At almost every frequency point, TPG of HF Broadband Fan Antenna matching network increases, as $\lambda$ decreases.

2) At almost every frequency point, VSWR of HF Broadband Fan Antenna matching network increases, as $\lambda$ decreases.

3) From Table 6-4, if the design constrain is to have VSWR < 3, the matching network with $\lambda$=0.6 is the best choice. VSWR of the fan antenna before and after matching is shown in Figure 6.40.
Chapter 7

Conclusion

7.1 Dissertation Contributions

This dissertation has two main contributions. The first is to evaluate four nature-inspired optimization algorithms (GA, PSO, ACO and FA), of which two (ACO and FA) are relatively new for the antenna design research area. The test problem was the design of an impedance matching network for an 80-meter dipole antenna. The objective was to minimize the maximum VSWR over the frequency range of 3.5MHz to 3.85MHz.

Evaluations of these four nature-inspired algorithms were performed by comparing them both with the global optimization toolbox from the MATLAB software package and among themselves. The optimization of these four nature-inspired algorithm by MATLAB-based codes showed a better result than the global optimization toolbox. The main focuses for evaluation were their accuracy and speed. The four nature-inspired algorithms achieved the more accurate and faster speed: specifically, the FA showed the best results overall.

The second contribution of this dissertation is improving the FA, which had the best result from the first evaluation and also has a special ability in that it can find multiple local optimal points at a given time, which can be useful in many multi-optima problems. A new modified firefly algorithm (MFA) has been invented and written along with the guidelines for parameter selection. The MFA modifies the SFA in three points:

1) Scaling a computation domain into a unit hypercube
2) Having a realistic attractiveness between fireflies
3) Including a new form of absorption coefficient and a random parameter
The performance of the proposed algorithm was tested in finding optimal solutions for eleven standard benchmark functions. Results have shown that its performance in obtaining an optimal solution for all functions is better compared to SFA in terms of finding an optimal value closer to the optimal point and doing so more quickly. Moreover, MFA was used to design a lossless impedance matching network and lossy impedance matching network and yielded results that were more accurate and faster than SFA.

MFA is proven to be a good alternative solution for optimizing various problems including the antenna impedance matching problem. In addition, because of the new parameter selection guidelines, only two parameters are required for fine-tuning. Therefore, it does not require the more complicated SFA or other nature-inspired implementations.
7.2 Future Research Directions

In the last decade, many new nature-inspired algorithms have been invented, including the artificial bee colony algorithm, social spider algorithm, bat algorithm, strawberry algorithm, plant propagation algorithm, seed-based plant propagation algorithm, and many more. Some of these are effective compared to other early algorithms; however, most of them have not been studied extensively, especially in the antenna design research field.

Due to the results of many test problems in this dissertation, the proposed algorithm, MFA, is proven to be capable of finding the impedance matching network for an antenna in both lossless and lossy networks. In the future, more tests such as optimizing various types of antenna design problem can be done. These design problems are more complicated than the impedance matching problem chosen in this thesis, with more variables and objective values. However, because MFA is relatively easy to implement, with only a few parameters to adjust, it should be one of the most promising solutions. MFA can also potentially be utilized in applications outside of the antenna design research field such as signal processing system, linear control, and computer science.
Bibliography


[69] Introduction to Evolutionary Computing MIT Press Sept 2004


P. Srivatsava, B. Mallikarjun, and X. Yang, "Optimal test sequence generation using firefly algorithm," Swarm and Evolutionary Computation, 8, pp.44--53. 2013


J. L. Rodriguez, I. Garcia-Tunon, J. M. taboada, and F. O. Basteiro, Broadband HF Antenna Matching Network Design using a Real--coded Genetic Algorithm, IEEE Trans. on Antenna. And propagation, vol.55, N0 .3 March 2.007
Appendix A

Dipole Antenna Impedance File (A15.txt)

<table>
<thead>
<tr>
<th>Frequency(MHz)</th>
<th>Tag</th>
<th>Seg</th>
<th>Real(Z)</th>
<th>Imag(Z)</th>
<th>Mag(Z)</th>
<th>Phase(Z)</th>
<th>$Z_0$</th>
<th>VSWR</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.500</td>
<td>7</td>
<td>41</td>
<td>76.947</td>
<td>-78.021</td>
<td>109.582</td>
<td>-45.397</td>
<td>50.000</td>
<td>3.48</td>
</tr>
<tr>
<td>3.525</td>
<td>7</td>
<td>41</td>
<td>78.353</td>
<td>-67.071</td>
<td>103.139</td>
<td>-40.564</td>
<td>50.000</td>
<td>3.02</td>
</tr>
<tr>
<td>3.550</td>
<td>7</td>
<td>41</td>
<td>79.772</td>
<td>-56.143</td>
<td>97.548</td>
<td>-35.137</td>
<td>50.000</td>
<td>2.63</td>
</tr>
<tr>
<td>3.575</td>
<td>7</td>
<td>41</td>
<td>81.203</td>
<td>-45.231</td>
<td>92.951</td>
<td>-29.118</td>
<td>50.000</td>
<td>2.31</td>
</tr>
<tr>
<td>3.600</td>
<td>7</td>
<td>41</td>
<td>82.647</td>
<td>-34.331</td>
<td>89.494</td>
<td>-22.558</td>
<td>50.000</td>
<td>2.06</td>
</tr>
<tr>
<td>3.625</td>
<td>7</td>
<td>41</td>
<td>84.103</td>
<td>-23.442</td>
<td>87.309</td>
<td>-15.575</td>
<td>50.000</td>
<td>1.87</td>
</tr>
<tr>
<td>3.650</td>
<td>7</td>
<td>41</td>
<td>85.572</td>
<td>-12.560</td>
<td>86.489</td>
<td>-8.350</td>
<td>50.000</td>
<td>1.77</td>
</tr>
<tr>
<td>3.675</td>
<td>7</td>
<td>41</td>
<td>87.054</td>
<td>-1.681</td>
<td>87.070</td>
<td>-1.106</td>
<td>50.000</td>
<td>1.74</td>
</tr>
<tr>
<td>3.700</td>
<td>7</td>
<td>41</td>
<td>88.549</td>
<td>9.197</td>
<td>89.025</td>
<td>5.929</td>
<td>50.000</td>
<td>1.80</td>
</tr>
<tr>
<td>3.725</td>
<td>7</td>
<td>41</td>
<td>90.057</td>
<td>20.077</td>
<td>92.268</td>
<td>12.568</td>
<td>50.000</td>
<td>1.93</td>
</tr>
<tr>
<td>3.750</td>
<td>7</td>
<td>41</td>
<td>91.583</td>
<td>30.963</td>
<td>96.675</td>
<td>18.680</td>
<td>50.000</td>
<td>2.11</td>
</tr>
<tr>
<td>3.775</td>
<td>7</td>
<td>41</td>
<td>93.120</td>
<td>41.859</td>
<td>102.096</td>
<td>24.205</td>
<td>50.000</td>
<td>2.35</td>
</tr>
<tr>
<td>3.800</td>
<td>7</td>
<td>41</td>
<td>94.763</td>
<td>52.766</td>
<td>108.385</td>
<td>29.133</td>
<td>50.000</td>
<td>2.63</td>
</tr>
<tr>
<td>3.825</td>
<td>7</td>
<td>41</td>
<td>96.241</td>
<td>63.690</td>
<td>115.406</td>
<td>33.496</td>
<td>50.000</td>
<td>2.95</td>
</tr>
<tr>
<td>3.850</td>
<td>7</td>
<td>41</td>
<td>97.825</td>
<td>74.631</td>
<td>123.043</td>
<td>37.340</td>
<td>50.000</td>
<td>3.30</td>
</tr>
</tbody>
</table>
Appendix B

Fan antenna Impedance File (fan_ant.29.txt)

<table>
<thead>
<tr>
<th>Freq(MHz)</th>
<th>Real(Z)</th>
<th>Imag(Z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.000</td>
<td>27.27</td>
<td>-27.27</td>
</tr>
<tr>
<td>2.375</td>
<td>54.54</td>
<td>54.54</td>
</tr>
<tr>
<td>2.750</td>
<td>200</td>
<td>109.08</td>
</tr>
<tr>
<td>3.125</td>
<td>136.35</td>
<td>0</td>
</tr>
<tr>
<td>3.500</td>
<td>172.72</td>
<td>54.54</td>
</tr>
<tr>
<td>4.250</td>
<td>254.54</td>
<td>0</td>
</tr>
<tr>
<td>5.000</td>
<td>200</td>
<td>-109.08</td>
</tr>
<tr>
<td>5.750</td>
<td>118.18</td>
<td>-145.44</td>
</tr>
<tr>
<td>6.500</td>
<td>72.72</td>
<td>-90.90</td>
</tr>
<tr>
<td>7.250</td>
<td>54.54</td>
<td>-54.54</td>
</tr>
<tr>
<td>8.000</td>
<td>36.36</td>
<td>-27.27</td>
</tr>
<tr>
<td>8.750</td>
<td>36.36</td>
<td>0</td>
</tr>
<tr>
<td>9.500</td>
<td>36.36</td>
<td>18.18</td>
</tr>
<tr>
<td>10.00</td>
<td>50.0</td>
<td>36.36</td>
</tr>
<tr>
<td>10.25</td>
<td>81.81</td>
<td>54.54</td>
</tr>
<tr>
<td>11.00</td>
<td>90.90</td>
<td>81.81</td>
</tr>
<tr>
<td>11.76</td>
<td>127.26</td>
<td>90.90</td>
</tr>
<tr>
<td>12.50</td>
<td>163.62</td>
<td>18.18</td>
</tr>
<tr>
<td>13.25</td>
<td>127.26</td>
<td>-18.18</td>
</tr>
<tr>
<td>14.00</td>
<td>118.18</td>
<td>-18.18</td>
</tr>
<tr>
<td>14.75</td>
<td>81.81</td>
<td>0</td>
</tr>
<tr>
<td>15.00</td>
<td>81.81</td>
<td>18.18</td>
</tr>
<tr>
<td>15.50</td>
<td>81.81</td>
<td>36.36</td>
</tr>
<tr>
<td>16.25</td>
<td>81.81</td>
<td>81.81</td>
</tr>
<tr>
<td>17.00</td>
<td>81.81</td>
<td>90.90</td>
</tr>
<tr>
<td>17.75</td>
<td>86.35</td>
<td>109.08</td>
</tr>
<tr>
<td>18.50</td>
<td>86.35</td>
<td>122.715</td>
</tr>
<tr>
<td>19.25</td>
<td>99.99</td>
<td>154.53</td>
</tr>
<tr>
<td>20.00</td>
<td>163.63</td>
<td>163.63</td>
</tr>
</tbody>
</table>
Appendix C

MATLAB-based Code for GA

clear all

global A15 freq real imag omega sol99 gen sfmin

importfile('A15.txt');

range=[1e-9 8e-9 1e-9 8e-9];

Ngrid=100;

dx=(range(2)-range(1))/Ngrid;

dy=(range(4)-range(3))/Ngrid;

[x,y]=meshgrid(range(1):dx:range(2),... 
    range(3):dy:range(4));

    x1(3)= 0.68711e-6;

n1=length(x);

for i1=1:n1
    for i2=1:n1
        x1(1)=y(i2,i1);
        x1(2)=x(i2,i1);
        z(i1,i2)= 1/objfun_ga(x1);
    end
end

xx=x;

yy=y;

tic

%% Basic GA parameters

gaDat.Objfun='objfun_ga';
lb=[1e-9 1e-9 500e-9];
ub=[8e-9 8e-9 900e-9];

\[\text{gaDat.FieldD}=[\text{lb}; \text{ub}];\]

% Execute GA
\[\text{gaDat}=\text{ga_nov9(gaDat)};\]

% Result are in
\[\text{solution} = \text{gaDat.xmin}\]
\[\text{f} \_\text{min} = \text{gaDat.fxmin}\]
\[\text{toc}\]
\[\text{c1}=\text{solution}(1);\]
\[\text{c2}=\text{solution}(2);\]
\[\text{l1}=\text{solution}(3);\]

%freq = A15(:,1).*1e6; % frequency in MHz
%real = A15(:,4);
%imag = A15(:,5);
%omega = 2*pi.*freq;

\[\text{for m}=1:15\]
\[\text{zload(m)} = \text{real(m)} + 1i*\text{imag(m)};\]
\[\text{xc1t} = 1/(\text{omega(m)}*\text{c1});\]
\[\text{xc2t} = 1/(\text{omega(m)}*\text{c2});\]
\[\text{xlt} = \text{omega(m)}*\text{l1};\]
\[\text{x} = [\text{xc1t,xc2t,xlt}];\]
\[\text{g(m)} = \text{Pi3(x,zload(m))};\]
\[\text{end}\]

Min_VSWR = min(g)
Max_VSWR = max(g)
figure(1);
plot(freq,g);
xlabel('Frequency/Hz');
ylabel('VSWR');
title('VSWR with matching network optimized');
figure(2);
contour(xx,yy,z,15);
hold on
for i=1:gen
    plot(sol99(i,1),sol99(i,2),'r*')
    axis([1e-9 8e-9 1e-9 8e-9]);
    xlabel('c1/Frad');
ylabel('c2/Frad');
    writerObj = VideoWriter('out.avi'); % Name it.
writerObj.FrameRate = 10; % How many frames per second.
open(writerObj);
frame(i) = getframe(gcf); % 'gcf' can handle if you zoom in to take a movie.
writeVideo(writerObj, frame);
end
figure(3);
x=1:gen;
y=sfmin;
plot(x,y);
xlabel('# of generation');
ylabel('fmin');
function value=objcfun_ga(x)
global A15 freq real imag omega

%y = 100*(x(1)^2 - x(2))^3 + (1 - x(3))^2 + abs(0.4 - x(2));

freq = A15(:,1).*1e6; % frequency in MHz
real = A15(:,4);
imag = A15(:,5);
omega = 2*pi.*freq;
%start = [2e-9 5e-9 6e-7]; % setting start point
zs=50;
for j=1:15
zload(j) = real(j) + 1i*imag(j);
xc1(j) = 1/(omega(j)*x(1));
xc2(j) = 1/(omega(j)*x(2));
xl(j) = omega(j)*x(3);
x0 = [xc1(j) xc2(j) xl(j)];
g(j) = Pi3(x0,zload(j));
end
[Vmax,k] = max(g);
% calculate VSWR
value=Vmax;
Appendix D

MATLAB-based Code for PSO

%% Particle Swarm Optimization Simulation

%% Initialization

clear all
clc

global A15

tic

importfile('A15.txt')

up = [5e-9 9e-9 800e-9];

lp = [1e-9 5e-9 500e-9 ];

grd=100;

n = 70;       % Size of the swarm "no of birds"

bird_setp = 150; % Maximum number of "birds steps"

dim = 3;      % Dimension of the problem

C2 =1.2;      % PSO parameter C1

C1 = 1.2;     % PSO parameter C2

w =0.9;       % pso momentum or inertia

fitness= 3*ones(n,bird_setp);

R1 = rand(dim, n);

R2 = rand(dim, n);

current_fitness = 3*ones(n,1);

% Initializing swarm and velocities and position %

for i=1:n

    current_position(1,i)= lp(1) +(up(1)-lp(1))*rand;

current_position(2,i) = lp(2) + (up(2)-lp(2))*rand;
current_position(3,i) = lp(3) + (up(3)-lp(3))*rand;
end

velocity = 0.3*randn(dim, n);
velocity = 1e-6*randn(dim, n); % scale velocity by 8e-6

current_position = 3e-9*(rand(dim, n)-1);
velocity = 3e-8*randn(dim, n);

local_best_position = current_position;

for i = 1:n
    current_fitness(i) = Live_fn(current_position(:,i));
end

local_best_fitness = current_fitness;

[global_best_fitness, g] = min(local_best_fitness);
for i = 1:n
    globl_best_position(:,i) = local_best_position(:,g);
end

velocity = w *velocity + c1*(R1.*(local_best_position-current_position)) +
c2*(R2.*(globl_best_position-current_position));
current_position = current_position + velocity;

%% Main Loop

iter = 0; % Iterations counter
while ( iter < bird_setp )
    iter = iter + 1;
    for i = 1:n,
        current_fitness(i) = Live_fn(current_position(:,i));
    end
end
end

for i = 1 : n
    if current_fitness(i) < local_best_fitness(i)
        local_best_fitness(i) = current_fitness(i);
        local_best_position(:,i) = current_position(:,i) ;
    end
end

[current_global_best_fitness,g] = min(local_best_fitness);
if current_global_best_fitness < global_best_fitness
    global_best_fitness = current_global_best_fitness;
    for i=1:n
        globl_best_position(:,i) = local_best_position(:,g);
    end
end

velocity = w *velocity + c1*(R1.*(local_best_position-current_position)) +
c2*(R2.*(globl_best_position-current_position));
current_position = current_position + velocity;
end % end of while loop its mean the end of all step that the birds move it
toc

[Jbest_min,I] = min(current_fitness) % minimum fitness

    current_position(:,I) % best solution

function z=Live_fn(x)
    % Optimize PI3 network
    global A15
    freq = A15(:,1).*1e6; % frequency in MHz
    real = A15(:,4);
    imag = A15(:,5);
omega = 2*pi.*freq;

% start = [2e-9 5e-9 6e-7]; % setting start point
zs=50;

% set source impedance
for j=1:15
zload(j) = real(j) + 1i*imag(j);
xc1(j) = 1/(omega(j)*x(1));
xc2(j) = 1/(omega(j)*x(2));
xl(j) = omega(j)*x(3);
x0 = [xc1(j) xc2(j) xl(j)];
g(j) = Pi3(x0,zload(j));
end
[Vmax,k] = max(g);
Appendix E

MATLAB-based Code for ACO

clear all;
global A 15 freq real imag omega
importfile('A15.txt')
aco1
function result = aco1();
global A 15 freq real imag omega
Ngrid=100;
range=[1e-9 8e-9 1e-9 8e-9];
dx=(range(2)-range(1))/Ngrid;
dy=(range(4)-range(3))/Ngrid;
xx=range(1):dx:range(2);
yy=range(3):dy:range(4);
[x,y]=meshgrid(range(1):dx:range(2),...
  range(3):dy:range(4));
x1(3)= 0.7194e-6;
n1=length(xx);
n2=length(yy);
for i1=1:n2
  for i2=1:n1
    x1(1)=x(i1,i2);
    x1(2)=y(i1,i2);
    z(i1,i2)= 1/dp1(x1);
  end
end

nVar = 3;
nSize = 100; %size
nAnts = 40;
%nAnts = 2;
fopt = 1.34; %Apriori optimal
q=0.1;
qk=q*nSize;
xi = 0.85;
maxiter = 200;
%maxiter = 2000;
errormin = 1e-05;
%errormin = 1e-04; %Stopping criteria
%Parameter range
Up = 8e-7*ones(1,nVar); %range for dp function
Lo = 1e-9*ones(1,nVar);
S = zeros(nSize,nVar+1,1);
Solution = zeros(nSize,nVar+2,1);
for k=1:nSize
Srand = zeros(nVar);
for j = 1:nAnts
for i=1:nVar
Srand(j,i) = (Up(i) - Lo(i))* rand(1) + Lo(i); %uniform distribution
end
ffbest(j)=dp1(Srand(j,:)); %dp test function
end
end
[fbest kbest] = min(FFbest);
S(k,:)=[Srands(kbest,:) fbest];
end

% Rank the archive table from the best (the lowest)
S = sortrows(S,nVar+1);
w = zeros(1,nSize);
for i=1:nSize
    w(i) = pdf('Normal',i,1.0,qk);
end
Solution=S;

% End of archive table initialization
stag = 0;

% Iterative process
figure(1);
for iteration = 1: maxiter
    p=w/sum(w);
    ref_point = mean(Solution(:,nVar+1));
    for i=1:nSize
        pw(i) = weight_prob(p(i),0.6);
        objv(i)= valuefunction(0.8,0.8, 2.25, ref_point-Solution(i, nVar+1));
        prospect(i) = pw(i)*objv(i);
    end
    [max_prospect ix_prospect]=max(prospect);
    selection = ix_prospect;
    % Phase two, calculate Gi
    % First calculate standard deviation
delta_sum=zeros(1,nVar);
for i=1:nVar
for j=1:nSize
delta_sum(i) = delta_sum(i) + abs(Solution(j,i) - Solution(selection,i)); %selection
end
delta(i)=xi /(nSize - 1) * delta_sum(i);
end
Stemp = zeros(nAnts,nVar);
for k=1:nAnts
for i=1:nVar
Stemp(k,i) = delta(i) * randn(1) + Solution(selection,i); %selection
if Stemp(k,i) > Up(i)
Stemp(k,i) = Up(i);
elseif Stemp(k,i) < Lo(i);
Stemp(k,i) = Lo(i);
end
end
end
ffeval(k) =dp1(Stemp(k,:)); %dp test function
end
for ii =1:nAnts
xn(ii)=Stemp(ii,1);
yn(ii)= Stemp(ii,2);
end
contour(x,y,z,15);
hold on
plot(xn,yn,'.','markersize',10,'markerfacecolor','g');
axis([1e-9 8e-9 1e-9 8e-9]);
xlabel('c1(Farad)');
ylabel('c2(Farad)');
title('c1 vs c2');

writerObj = VideoWriter('out.avi'); % Name it.
writerObj.FrameRate = 10; % How many frames per second.
open(writerObj);
pause(.1)
frame(iteration) = getframe(gcf); % 'gcf' can handle if you zoom in to take a movie.
writeVideo(writerObj, frame);
hold off
Ssample = [Stemp ffeval']; % put weight zero
% insert this solution to archive, all solution from ants
Solution_temp = [Solution; Ssample];
% sort the solution
Solution_temp = sortrows(Solution_temp,nVar+1);
% remove the worst
Solution_temp(nSize+1:nSize+nAnts,:)=[];
Solution = Solution_temp ;
best_par(iteration,:) = Solution(1,1:nVar);
best_obj(iteration) = Solution(1,nVar+1);
% check stopping criteria
if iteration > 1
    dis = best_obj(iteration-1) - best_obj(iteration);
    if dis <= 1e-04
stag = stag + 1;
else
stag = 0;
end
end

ftest = Solution(1,nVar+1);

%Solution(1,1:nVar);
if abs(ftest - fopt) < errormin || stag >= 5000
break
end
end

c1=Solution(1,1);
c2=Solution(1,2);
l1=Solution(1,3);
for m=1:15
zload(m) = real(m) + 1i*imag(m);
xc1t = 1/(omega(m)*c1);
xc2t = 1/(omega(m)*c2);
xlt = omega(m)*l1;
x = [xc1t,xc2t,xlt];
g(m) = Pi3(x,zload(m));
end
Min_VSWR = min(g)
Max_VSWR = max(g)
figure(2);
plot(freq,g);
xlabel('Frequency/Hz');
ylabel('VSWR');
title('VSWR with matching network optimized');

function value = valuefunction(alpha, beta, lambda, xinput)
value = zeros(1, length(xinput));
for i = 1:length(xinput)
    if xinput(1,i) >= 0
        value(1,i) = xinput(1,i) ^ alpha;
    else
        value(1,i) = -lambda * (-xinput(1,i))^ beta;
    end
end

function prob = weight_prob(x, gamma)
% weighted the probability
% gamma is weighted parameter
prob = zeros(1, length(x));
for i = 1:length(x)
    if x(i) < 1
        prob(i) = (x(i)^gamma) / (((x(i)^gamma) + (1-x(i))^gamma)^(1/gamma));
    else
        prob(i) = 1.0;
    end
end
Appendix F

MATLAB-based Code for FA

% Firefly algorithm
function [best]=firefly_simple1(instr)

% n=number of fireflies
% MaxGeneration=number of pseudo time steps
if nargin<1, instr=[50 50]; end
n=instr(1); MaxGeneration=instr(2);
rand('state',0); % Reset the random generator
range=[-5 5 -5 5];
alpha=0.2; % Randomness 0--1 (highly random)
gamma=1.0; % Absorption coefficient
delta=0.97; % Randomness reduction (similar to
% an annealing schedule)

% Grid values are used for display only
Ngrid=100;
dx=(range(2)-range(1))/Ngrid;
dy=(range(4)-range(3))/Ngrid;
[x,y]=meshgrid(range(1):dx:range(2),...%range(3):dy:range(4));
z=fa1(x,y);

% Display the shape of the objective function
figure(1); surfc(x,y,z);

% -------------------------------------------------
% generating the initial locations of n fireflies

[zn,yn,Lightn]=init_ffa(n,range);

% Display the paths of fireflies in a figure with
% contours of the function to be optimized

figure(2);

% Iterations or pseudo time marching

for i=1:MaxGeneration, %%% start iterations

% Show the contours of the function

contour(x,y,z,15); hold on;

% Evaluate new solutions

zn=fa1(xn,yn);

% Ranking the fireflies by their light intensity

[Lightn,Index]=sort(zn);

xn=xn(Index); yn=yn(Index);

xo=xn; yo=yn; Lighto=Lightn;

% Trace the paths of all roaming fireflies

plot(xn,yn,'.','markersize',10,'markerfacecolor','g');

% Move all fireflies to the better locations

[zn,yn]=ffa_move(xn,yn,Lightn,xo,yo,Lighto,alpha,gamma,range);

drawnow;

% Use "hold on" to show the paths of fireflies

hold off;

% Reduce randomness as iterations proceed

alpha=newalpha(alpha,delta);

end   %%% end of iterations

best(:,1)=xo;
best(:,2)=yo';
best(:,3)=Lighto';

% ----- All subfunctions are listed here -------

% The initial locations of n fireflies
function [xn,yn,Lightn]=init_ffa(n,range)
xrange=range(2)-range(1);
yrange=range(4)-range(3);
xn=rand(1,n)*xrange+range(1);
yn=rand(1,n)*yrange+range(3);
Lightn=zeros(size(yn));

% Move all fireflies toward brighter ones
function [xn,yn]=ffa_move(xn,yn,Lightn,xo,yo,...
    Lighto,alpha,gamma,range)
ni=size(yn,2); nj=size(yo,2);
for i=1:ni,
    % The attractiveness parameter beta=exp(-gamma*r)
    for j=1:nj,
        r=sqrt((xn(i)-xo(j))^2+(yn(i)-yo(j))^2);
        if Lightn(i)<Lighto(j), % Brighter and more attractive
            beta0=1; beta=beta0*exp(-gamma*r.^2);
            xn(i)=xn(i).*((1-beta)+xo(j).*beta+alpha.*(rand-0.5));
            yn(i)=yn(i).*((1-beta)+yo(j).*beta+alpha.*(rand-0.5));
        end
    end % end for j
end % end for i

[xn,yn]=findrange(xn,yn,range);
% Reduce the randomness during iterations
function alpha=newalpha(alpha,delta)
alpha=alpha*delta;

% Make sure the fireflies are within the range
function [xn,yn]=findrange(xn,yn,range)
for i=1:length(yn),
    if xn(i)<=range(1), xn(i)=range(1); end
    if xn(i)>=range(2), xn(i)=range(2); end
    if yn(i)<=range(3), yn(i)=range(3); end
    if yn(i)>=range(4), yn(i)=range(4); end
end
%====================================

function z=fa1(x,y)
z= exp(-(x-4).^2-(y-4).^2)+exp(-(x+4).^2-(y-4).^2)+2*exp(-x.^2-(y+4).^2)+2*exp(-x.^2-y.^2);
% z= ( abs(x)+abs(y)).*exp( - x.^2 - y.^2);
Appendix G

MATLAB-based Code for MFA

% Main program of Modified Firefly Algorithm (MFA) Maximization problem
% For minimization of function ,f, it is equivalent to maximize -f(x)

global A15 A25 freq real1 imag1 omega ffa

global gm theta1 theta2 count lamda

importfile('A25.txt');
importfile('A15.txt');

% start count number of function calls

count=0;

tic
%clear all;clc

%input

% N: Number of population.
% max_it: Maximum number of iterations (t).
% F_index: The index of the test function.
% gamma : distance scale parameter
% alpha : random parameter

%outputs:

% Fbest: Best result.
% Lbest: Best solution. The location of Fbest in search space.
% BestChart: The best so far Chart over iterations.
% MeanChart: The average fitnesses Chart over iterations.

gamma = input(' input gamma i.e 1-200   ')

% gm is a gamma value used in MFA
gm=gamma;

alpha = input(' input alpha i.e. 0.01 - 0.8  ')

F_index=input(' input function number i.e. 1-40  ')

ffa=input(' input: 0 for fa, OR 1 for MFA  ')

% for lossy ip3 network (F_index= 38) only, lambda is needed
lamda = input(' input lamda i.e 0-1  ')

[low,up,dim]= test_functions_range(F_index);

% Formula for maximum number of iteration
max_it= 20*dim +10;

max_it= 40;

% theta1 is in the formula : gamma= gamma0*theta1^t
cc1= (log(1e-2)/max_it);
theta1= exp(cc1);

% theta2 is in the formula : Alpha= Alpha0*theta2^t
cc2= (log(1e-15/alpha)/max_it);
theta2= exp(cc2);

% Formula for number of population
N=5*dim +40;

[Fbest,Lbest,BestChart,MeanChart]=mfa(F_index,N,max_it,gamma,alpha);

% display the maximum objective function value
Fbest

    for j=1:dim
        Lbest(j)= (up(j)-low(j))*Lbest(j) + low(j);
    end

% display the best solution
Lbest
% display number of function count
fn_call = count

% display execution time
toc

% plot Best so far cost vs. iteration
semilogy(BestChart,'--k');
title(['\textbf{F',num2str(F_index,1)}]);
xlabel('\textbf{Iteration}');ylabel('\textbf{Best Cost}');
legend('\textbf{MFA}',1);

function [Fbest,Lbest,BestChart,MeanChart]=mfa(F_index,N,max_it,gamma,alpha)
global theta
theta1 theta2 gm count

% M: function value
% dim: Dimension of the test function.
% N: Number of population.
% R: Distance between agents in search space.
% [low-up]: Allowable range for search space.
% get allowable range and dimension of the test function.
[low,up,dim]= test_functions_range(F_index);

% random initialization for agents.
X=initialization(dim,N);

% create the best so far chart
BestChart=[];

% Begin iterations
for iteration=1:max_it
% Checking allowable range.

X = space_bound(X);

% Transform to the original working space
for i = 1:N
    for j = 1:dim
        X1(i,j) = (up(j)-low(j))*X(i,j) + low(j);
    end
end

% Evaluate function value for each firefly position
fitness = evaluateF(X1, F_index);
[best best_X] = max(fitness); % maximization.

if iteration == 1
    Fbest = best; Lbest = X(best_X,:);
end

if best > Fbest % maximization
    Fbest = best; Lbest = X(best_X,:);
end

BestChart = [BestChart Fbest];
MeanChart = [MeanChart mean(fitness)];

[M] = IntensityCalculation(fitness);

% populations movement.
[X] = move(X, M, gamma, alpha, fitness, low, up, F_index);
alpha = alpha * theta2;
gm = gm * theta1;
end % iteration

function fitness = evaluateF(X, F_index)

% This function finds a function value for each firefly position
global count

[N,dim]=size(X);

for i=1:N
    %L is the location of population number 'i'
    L=X(i,:);
    %calculation of objective function for population number 'i'
    fitness(i)=test_functions(L,F_index,dim);
    count =count+1;
end

function [X]=initialization(dim,N)
    % initialize position of each firefly
    X=rand(N,dim);
end

function [M]=IntensityCalculation(fit)
    % normalize function value to 0 and 1
    Fmax=max(fit); Fmin=min(fit); Fmean=mean(fit);
    [i N]=size(fit);
    if Fmax==Fmin
        M=ones(1,N);
    else
        best=Fmax;worst=Fmin;
        M=(fit-worst)./(best-worst);
    end
end

function [X]= move(X,M,gamma,alpha,fitness,low,up,F_index)
    % move firefly from position at iteration t to t+1
    global ffa gm
[N,dim]=size(X);
[M,index]= sort(M);
dmax=sqrt(dim);
X = X(index,:);
for i=1:N-1
  xtem=X(i,:);
  for j=i+1:N
    Rij=norm(X(i,:)-X(j,:),2)/dmax ;
    if ffa == 0
      bta=1*exp(-gamma*Rij^2);
    else
      bta= 1*M(j)*exp(-gm*Rij^2);
    end
    newx= X(i,:)+ bta.*(X(j,:)-X(i,:))+ alpha.*(rand(1,dim)-0.5);
    for jj=1:dim
      L(jj)= (up(jj)-low(jj))*newx(jj) + low(jj);
    end
    fit1=test_functions(L,F_index,dim);
    if fit1 > fitness(i)
      xtem=newx;
      fitness(i)= fit1;
    end
  end
end
X(i,:)=xtem;
end
function X=space_bound(X)
% Limit variable value between 0 an 1

[N,dim]=size(X);

for i=1:N
    for j=1:dim
        if X(i,j) < 0
            X(i,j) = 0.000001;
        end
        if X(i,j) > 1
            X(i,j) = 0.999999;
        end
    end
end

function m = Pi3(x,zload)

% This function computes VSWR of the lossless PI3 network
zs = 50;

% calculate input impedance looking from source side
para = 1i*x(3)*zload/(1i*x(3)+zload);
seri = para - 1i*x(2);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);

% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);

% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end

function m = Pi4(zs,x,zload)

% calculate VSWR of Fano LCR circuit with zs is constant
% calculate input impedance looking from source side
para = (-1i*x(2)*zload)/(-1i*x(2)+zload);
seri = para +1i*x(3);
zin = -seri*1i*x(1)/(-1i*x(1)+seri);
% calculate reflection coefficient
gamma = (zin-zs)/(zin+zs);
% calculate VSWR
m = (1+abs(gamma))/(1-abs(gamma));
end
Appendix H

MATLAB codes for testing functions

function fit=test_functions(L,F_index,dim)

% objective function with F_index, L is a dim-dimension variable.

if F_index==1
  % sphere or Dejong function
  fit=-sum((L).^2);
end

if F_index==2
  % function from Yang's paper
  fit= exp(-(L(1)-4)^2-(L(2)-4)^2)+exp(-(L(1)+4)^2-(L(2)+4)^2)+2*exp(-(L(1)^2-(L(2)-4)^2)+2*exp(-L(1)^2-L(2)^2));
end

if F_index==3
  % four peaks fn. (use gamma=400, c99=0.545, alpha=0.015, delta=0.97)
  fit= ( abs(L(1))+abs(L(2)))*exp(- L(1)^2 - L(2)^2);
end

if F_index==4
  % Ackley function
  fit= 20*exp(-0.2*sqrt(L(1)^2 + 0.5*L(2)^2)) +exp(0.5*cos(2*pi*L(1))+ 0.5*cos(2*pi*L(2)))-20-exp(1);
end

% de jong's function (sphere)

% z= -x.^2 - y.^2;

if F_index==5
  % Yang's forest function
  fit= -( abs(L(1))+abs(L(2)))*exp(- sin(L(1)^2) - sin(L(2)^2));
end
if F_index== 6
% rastrigin's function
fit= -20 - (L(1)^2 -10*cos(2*pi*L(1))) -(L(2)^2 -10*cos(2*pi*L(2)));
end
if F_index== 7
% zakharov's function
fit= -(L(1)^2 -L(2)^2 - (0.5*L(1) + L(2))^2 - (0.5*L(1) + L(2))^4);
end
if F_index== 8
% michalewicz function
fit= sin(L(1))*(sin(L(1)^2/pi))^20 + sin(L(2))*(sin(2*L(2)^2/pi))^20;
end
if F_index== 9
% rosenbrock function
fit= -(1-L(1)) ^2 - 100*(L(2)-L(1)^2) ^2;
end
if F_index== 10
% ip3 circuit
global A15 A25 freq real1 imag1 omega
x=L;
% y = 100*(x(1)^2 - x(2))^3 + (1 - x(3))^2 + abs(0.4 - x(2));
freq = A15(:,1).*1e6; % frequency in MHz
real1 = A15(:,4);
imag1 = A15(:,5);
omega = 2*pi.*freq;
%start = [2e-9 5e-9 6e-7]; % setting start point
% set source impedance
zs=50;
for j=1:15
zload(j) = real1(j) + 1i*imag1(j);
xc1(j) = 1/(omega(j)*x(1));
xc2(j) = 1/(omega(j)*x(2));
xl(j) = omega(j)*x(3);
x0 = [xc1(j) xc2(j) xl(j)];
g(j) = Pi3(x0,zload(j));
end
[Vmax,k] = max(g);
fit=-Vmax;
end
if F_index== 11
  %fano load network (zs is not variable). find TPG from gamma,
  % Use min-max(TPG)
x=L;
zs=2.205;
% set source impedance (FANO load equalizer)
dw=1/21;
for j=1:21
  w= j*dw;
  zload(j) = 1/(1+(1.2*w)^2) + 1i*w*(2.3-1.2/(1+(1.2*w)^2));
  xc1(j) = 1/w*x(1);
  xc2(j) = 1/w*x(2);
  xl(j) = w*x(3);
x0 = [xc1(j) xc2(j) xl(j)];
g(j) = Pi4(zs,x0,zload(j));
end

[Vmax,k] = max(g);
% gamma= (Vmax-1)/(Vmax+1);
fit= -Vmax;
end
if F_index==12

% fano load network (zs is constant), Maximize Minimum of TPG , Maximize the min(TPG);
x=L;
zs=2.205;
% set source impedance (FANO load equalizer)
% fit=0;
dw=1/21;
c=1.2;
r=1;
L=2.3;
for j=1:21
    w= j*dw;
    zload = 1i*w*L + 1*(-1i/(w*c))/(1 -1i/(w*c));
    xc1 = -1/(w*x(1));
    xc2 = -1/(w*x(2));
    xL = w*x(3);
    z1=(1i*xc2*zload)/(zload + 1i*xc2);
    z2 =z1 + 1i*xL;
    zin= (1i*xc1*z2)/(1i*xc1+z2);
gamma = (zin-zs)/(zin+zs);
TPG(j) = 1 - abs(gamma)^2;
end
fit = min(TPG);
end
if F_index == 13
% lossy ip3 circuit, Min max of lamda*abs(\Gamma) + (1-lamda)*(1-TPG);
% Did not use scale variables
global A15 A25 freq real1 imag1 omega lamda
x=L;
freq = A15(:,1).*1e6; % frequency in MHz
real1 = A15(:,4);
imag1 = A15(:,5);
omega = 2*pi.*freq;
zs = 50;
for j=1:15
zl = real1(j) + 1i*imag1(j);
xc1 = 1/(omega(j)*x(1));
xc2 = 1/(omega(j)*x(2));
xl = omega(j)*x(3);
xr = x(4);
z1 = (1i*xl*zl)/(1i*xl+zl);
z2 = -1i*xc2 + z1;
z3 = (-1i*xc1*z2)/(-1i*xc1+z2);
zin = xr + z3;
T = (zin-50)/(zin+50);
\[ z_4 = z_s + x_r; \]
\[ z_5 = (-1i * x_c_1 * z_4) / (-1i * x_c_1 + z_4); \]
\[ z_6 = z_5 - 1i * x_c_2; \]
\[ z_q = 1i * x_l * z_6 / (1i * x_l + z_6); \]
\[ r_q = \text{real}(z_q); \]
\[ x_q = \text{imag}(z_q); \]
\[ r_{\text{load}} = \text{real}(z_l); \]
\[ x_{\text{load}} = \text{imag}(z_l); \]
\[ TPG = 4 * r_q * r_{\text{load}} / ( (r_q + r_{\text{load}})^2 + (x_q + x_{\text{load}})^2); \]
\[ g(j) = \lambda \cdot \text{abs}(T) + (1 - \lambda) \cdot (1 - TPG); \]

\text{end}

\[ [V_{\text{max}}, k] = \text{max}(g); \]
\[ \text{fit} = -V_{\text{max}}; \]
\text{end}

\text{if } F_{\text{index}} == 14

\% fano LCR load network (rs is constant), objective is to maximize the
\% min(TPG)
\text{x = L;}
\text{zs = 2.205;}

\% set source impedance (FANO load equalizer)
\text{TPG = zeros(1, 21);}
\text{dw = 1/21;}
\text{for } j = 1:21

\text{w = j * dw;}
\text{zload = 1 / (1 + (1.2 * w)^2) + 1i * w * (2.3 - 1.2 / (1 + (1.2 * w)^2));}
\text{xc1 = -1 / (w * x(1));}
xc2 = -1/(w*x(2));
xl= w*x(3);

xr= zs;

z1=(xr*li*xc1)/(xr + li*xc1);

z2 =z1 + li*xl;

zq= (z2*1i*xc2)/(z2 + 1i*xc2);

rq= real(zq);

xq= imag(zq);

rload=real(zload);

xload=imag(zload);

TPG(j)= 4*rq*rload/( (rq + rload)^2 + (xq + xload)^2);

end

fit= min(TPG);

end

if F_index== 15

%       lossy ip3 circuit, use scale variables, L= 1e-6L , C = 1e-12C
%       use Min max of lamda*abs(Γ) + (1-lamda)*(1-TPG);

global A15 A25 freq real1 imag1 omega lamda

x=L;

freq = A15(:,1).*1e+6; % frequency in MHz

real1 = A15(:,4);

imag1 = A15(:,5);

omega = 2*pi.*freq;

zs=50;

for j=1:15

zl = real1(j) + 1i*imag1(j);
xc1 = 1/(omega(j)*1e-12*x(1));
xc2 = 1/(omega(j)*1e-12*x(2));
xl = omega(j)*1e-9*x(3);
xr=x(4);
z1 = (1i*xl*zl)/(1i*xl+zl);
z2= -1i*xc2 + z1;
z3= (-1i*xc1*z2)/(-1i*xc1+z2);
zin = xr + z3;
T= (zin-50)/(zin+50);
z4= zs+xr;
z5 = (-1i*xc1*z4)/(-1i*xc1+z4);
z6= z5 -1i*xc2;
zq = 1i*xl*z6/( 1i*xl+z6 );
rq= real(zq);
xq= imag(zq);
rload=real(zl);
xload=imag(zl);
TPG= 4*rq*rload/ ( (rq + rload)^2 + (xq + xload)^2 );
g(j) =lamda*abs(T) + (1-lamda)*(1-TPG);
end
[Vmax,k] = max(g);
fit=-Vmax;
end
if F_index== 16
    X=L;
    % Styblinski-Tang function
% STYBLINSKYTANG([x1, x2, ..., xn]) returns the value of the
% Styblinski-Tang at the specified points. All [xi] may be vectors.
% The search domain is
% -5 < x_i < 5
% The global minimum is
% f(x1, x2, ..., xn) =
% f(-2.903534, -2.903534, ..., -2.903534) = -39.16599 * n
    fit = -sum(X.^4 - 16.*X.^2 + 5.*X)/2;
end
if F_index== 17
% Helical valley function
    x=L;
% The global minimum is
% f(x1, x2, x3) = f(1, 0, 0) = 0.
    fit = 100*((x(3) - 10*atan2(x(2), x(1))/pi)^2 + (sqrt(x(1)^2 + x(2)^2) - 1)^2) + x(3)^2;
    fit=-fit;
end
if F_index== 18
% Wood function
    x=L;
% The global minimum is
% f(x1, x2, x3, x4) = f(1, 1, 1) = 0
% output function value
    fit = 100*(x(1)^2 - x(2))^2 + (x(1) - 1)^2 + (x(3) - 1)^2 + 90*(x(3)^2 - x(4))^2 + ...
    10.1*((x(2) - 1)^2 + (x(4) - 1)^2) + 19.8*(x(2) - 1)*(x(4) - 1);
    fit=-fit;
if F_index== 19

% Powell's singular function
x=L;
%  The global minimum is
%  f(x1, x2, x3, x4) = f(0, 0, 0, 0) = 0.
% output function value

fit = (x(1) + 10*x(2))^2 + 5*(x(3) - x(4))^2 + (x(2) - 2*x(3))^4 + 10*(x(1) - x(4))^4;
fit=-fit;
end
if F_index== 20
% fan antenna circuit, use minimize max of lamda*abs(\Gamma) + (1-lamda)*(1-TPG);
global lamda  fan_ant
load fan_ant.txt
x=L;

freq = fan_ant(:,1);
% frequency in MHz
real1 = fan_ant(:,2);
imag1 = fan_ant(:,3);
omega = 2*pi.*freq;
zs=50;
for j=1:29
z1 = real1(j) + 1i*imag1(j);
x11 = (omega(j)*x(2));
x1 = -1/(omega(j)*1e-3*x(3));
x12 = omega(j)*x(4);
\[ xc_2 = -1/(\omega(j) \cdot 1e-3 \cdot x(5)); \]
\[ z_1 = z_2/(x(6)^2); \]
\[ z_2 = (1i \cdot x_2 \cdot z_1)/(1i \cdot x_2 + z_1); \]
\[ z_3 = (1i \cdot x_2 \cdot z_2)/(1i \cdot x_2 + z_2); \]
\[ z_4 = 1i \cdot x_1 + z_3; \]
\[ z_5 = (1i \cdot x_1 \cdot z_4)/(1i \cdot x_1 + z_4); \]
\[ z_{in} = 50 \cdot x(1) + z_5; \]
\[ T = (z_{in} - z_3)/(z_{in} + z_3); \]
\[ \text{za} = z_3 + 50 \cdot x(1); \]
\[ \text{zb} = (1i \cdot x_1 \cdot \text{za})/(1i \cdot x_1 + \text{za}); \]
\[ \text{zc} = 1i \cdot x_1 + \text{zb}; \]
\[ \text{zd} = (1i \cdot x_2 \cdot \text{zc})/(1i \cdot x_2 + \text{zc}); \]
\[ \text{ze} = (1i \cdot x_2 \cdot \text{zd})/(1i \cdot x_2 + \text{zd}); \]
\[ z_q = (x(6)^2) \cdot ze; \]
\[ rq = \text{real}(z_q); \]
\[ xq = \text{imag}(z_q); \]
\[ r_{load} = \text{real}(z_l); \]
\[ x_{load} = \text{imag}(z_l); \]
\[ \text{TPG} = 4 \cdot rq \cdot r_{load} / ( (rq + r_{load})^2 + (xq + x_{load})^2); \]
\[ g(j) = \lambda \cdot \text{abs}(T) + (1 - \lambda) \cdot (1 - \text{TPG}); \]
end

\[ [V_{max,k}] = \text{max}(g); \]
\[ \text{fit} = -V_{max}; \]
end
if F_{index} == 21
    \%anoload network (zs is constant), Maximize sum of TPG

x=L;
zs=2.205;

% set source impedance (FANO load equalizer)
%fit=0;
dw=1/21;
c=1.2;
r=1;
L=2.3;

for j=1:21
    w= j*dw;
    zload = 1i*w*L + 1*(-1i/(w*c))/(1 -1i/(w*c));
xc1 = -1/w*x(1);
xc2 = -1/w*x(2);
xL = w*x(3);
z1=(1i*xc2*zload)/(zload + 1i*xc2);
z2 =z1 + 1i*xL;
zin= (1i*xc1*z2)/(1i*xc1+z2);
gamma =(zin-zs)/(zin+zs);
TPG(j)= 1- abs(gamma)^2;
end
fit= sum(TPG);
end

function [down,up,dim]=test_functions_range(F_index)
% Give upper and lower bound of variables, and dimension of the function
if F_index==1
    dim=5;
end
for i =1:dim
    down(i)= -100;
    up(i)= 100;
end
end

if F_index== 2
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end

if F_index== 3
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end

if F_index== 4
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end
if F_index== 5
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end
if F_index== 6
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end
if F_index== 7
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end
if F_index== 8
    dim=2;
    for i =1:dim
        down(i)= 0;
        up(i)= pi;
end
end
if F_index== 9
    dim=2;
    for i =1:dim
        down(i)= -5;
        up(i)= 5;
    end
end
if F_index== 10
    dim=3;
    for i =1:2
        down(i)= 1e-9;
        up(i)= 0.8e-8;
    end
    down(3) = 0.1e-6;
    up(3)= 2e-6;
end
if F_index== 11
    dim=3;
    for i =1:3
        down(i)= 0.1;
        up(i)= 10;
    end
end
if F_index== 12
    dim=3;
    for i =1:3
        down(i)= 0.1;
        up(i)= 10;
    end
end

if F_index== 13
    dim=4;
    for i =1:2
        down(i)= 1e-9;
        up(i)= 0.8e-8;
    end
    down(3) = 0.1e-6;
    up(3)= 2e-6;
    down(4)= 1;
    up(4)= 1000;
end

    if F_index== 14
    dim=3;
    for i =1:3
        down(i)= 0.1;
        up(i)= 10;
    end
end

if F_index== 15
dim=4;
for i =1:2
    down(i)= 40;
    up(i)= 2000;
end
down(3) = 40;
up(3)= 3000;
    down(4)= 1;
up(4)= 1000;
end
    if F_index== 16
dim=3;
for i =1:3
    down(i)= -5;
    up(i)= 5;
end
    end
    if F_index== 17
dim=3;
for i =1:3
    down(i)= -5;
    up(i)= 5;
end
    end
    if F_index== 18
dim=4;

for i = 1:dim
    down(i) = -5;
    up(i) = 5;
end
end

if F_index == 19
    dim = 4;
    for i = 1:dim
        down(i) = -5;
        up(i) = 5;
    end
end

if F_index == 20
    dim = 6;
    down(1) = 0.001;
    down(2) = 1;
    down(3) = 0.1;
    down(4) = 1;
    down(5) = 0.1;
    down(6) = 0.001;
    up(1) = 1;
    up(2) = 12;
    up(3) = 2.5;
    up(4) = 12;
    up(5) = 2.5;
    up(6) = 5;
end

if F_index== 21
    dim=3;
    for i =1:3
        down(i)= 0.1;
        up(i)= 10;
    end
end
end
Vita

Nathawut Homsup

Nathawut Homsup was born in Bangkok, Thailand on December 9, 1987. He earned Bachelor of Science degree in Electrical Engineering from Chulalongkorn University in 2009. His Bachelor’s degree studies covered vast area of Electrical Engineering courses including Communication System, Electromagnetic Theory, Signal Processing, and Control System. His senior project title was “Vehicle Tracking by GPS and GSM”. Shortly after completing his undergraduate program, he attended graduate program at Electrical Engineering department of Pennsylvania State University. He received the Master of Science degree in 2011, with a paper topic “Performance Comparisons of Channel Coding Techniques for Digital Satellite Communications”, under supervision of Prof. John J. Metzner.

Working toward his Ph.D. study, he started working with Prof. James K. Breakall. His Ph.D. research focused on Antenna design optimization problem. His works were to compare and improve the optimization methods for antenna design. His dissertation topic was “Evaluation and Enhancement of Nature-Inspired Optimization for Antenna Impedance Matching”, which he used four nature-inspired optimization methods (GA, PSO, ACO, and FA) to optimize the impedance matching network design for 80 meters dipole antenna and compare them with the global optimization toolbox from MATLAB software. Also, he introduced new modified firefly algorithm (MFA), that was tested and compared with the standard firefly algorithm (SFA). The MFA showed better results than SFA in all benchmark test functions and impedance matching network design problems including both lossless and lossy networks.

During his graduate program at Pennsylvania State University, Nathawut published 1 conference papers and submitted 2 journal papers, all of which as first author. He is expected to graduate in December, 2016 and he can be reached via email nhu126@psu.edu or nathawut.homsup@gmail.com.