APPLICATION OF EVOLUTIONARY ALGORITHMS FOR OPTIMAL DIRECTIONAL OVERCURRENT RELAY COORDINATION

By

Stenane, Ndabeni Moses

Student Number: STNMOS001

Supervisor

Prof. Folly, KA

Komla.Folly@uct.ac.za

University of Cape Town, Department of Electrical Engineering, EBE Faculty

Submitted in fulfilment of the requirements for the degree of Master of Science in Electrical Engineering in the Department of Electrical Engineering at the University of Cape Town

Date: May 2014

Declaration

I, the undersigned, Ndabeni Moses Stenane, hereby declare that I know the meaning of plagiarism and that all the work in this thesis, except where properly acknowledged, is my original work.

Signed on this 27th day of May 2014

Signed by candidate Signature removed

Ndabeni Moses Stenane

Acknowledgements

I would like to thank, first and foremost, the God Almighty for granting me strength and wisdom during the course of my studies.

My utmost gratitude goes to my family- my wife Elizabeth, my mother Tlaleng, my father Velile, my sister Selloane, my brother Rasonti, my son Oratile and my daughter Ayanda for their support and words of courage when the task at hand seemed impossible.

I also want to convey my greatest appreciation to my supervisor, Professor Folly for his support and expert advice on the research. My special thanks are addressed to Farzad Razavi from Tafresh University for sharing knowledge from his previous work on the same area of research.

I also wish to thank all my friends and colleagues who stood by me throughout this journey. A special token of gratitude goes to my friend and colleague, Lesiba Mokgonyane for his support, assistance with ideas, modelling the power system network in the simulation tool and for assisting with the compilation of the code for conventional overcurrent relay coordination. I also wish to thank my friend and colleague, Taeza Snymes, for moral support and words of encouragement throughout the course of my studies. Special thanks also to Thabani Shibe at UCT who assisted with the printing and submission of the dissertation package.

I also wish to thank my sponsor Eskom Distribution, Free State OU, who made it possible with the financial support.

Abstract

Relay coordination is necessary to ensure that while protection relays operate as fast as possible, they are also able to isolate only the faulted parts of the system from the network, ensuring that a power system disturbance does not result in interruption of the power supply to a larger part of the power system network. Optimal relay coordination for overcurrent relays depends on two parameters, namely, Time Multiplier and Pickup Current Setting.

The conventional method of setting these two parameters for overcurrent relays applied on the power system network is to first determine the main and backup relay pairs which form part of the clockwise and anti-clockwise loops around the power system network. The relays are then set through an iterative process to ensure coordination. Initially, a general rule of setting relays to operate in 0.2 seconds for faults in the primary zone, to ensure fast operation, and in 0.2 seconds plus additional grading time, to ensure coordination, for faults in the backup zone is applied. The next relay in the loop is tested to check if it fulfils the requirements of the initial general rule. If the conditions of the general rule are not met, the previous relay's setting is adjusted to meet the requirements. This process is repeated until all the relays around the loop are set.

Conventional relay coordination process has a limitation in the sense that it is deterministic and the settings of subsequent relays depend on the initial guess of the settings of the initial relay. Therefore, this method does not necessarily provide solutions which guarantee optimal relay coordination but the best of the solutions tried.

Evolutionary Algorithms, due to their random nature and their ability to perform a parallel search for a number of potential solutions offer a possibility for optimal relay coordination.

In this research, a conventional overcurrent relay coordination method and a method of relay coordination through Evolutionary Algorithms were applied on the 230kV network of the IEEE 24 bus power system. For complex engineering problems such as relay coordination, optimal solutions to the problem are not known in advance. Therefore, the analysis of the performance of the applied methods was done through comparison.

The three Evolutionary Algorithms applied for overcurrent relay coordination in this research are: Genetic Algorithm (GA), Breeder Genetic Algorithm (BGA) and Population-Based Incremental Learning (PBIL). For PBIL, the effects of the learning rate on the algorithm were also investigated with the aim of improving the performance of the

algorithm. These Evolutionary algorithms were applied by formulating the problem of overcurrent relay coordination as an optimization problem. The main aim of a relay coordination study is to ensure that overcurrent relays operate as fast as possible for faults in the primary protection zone of the relays, while maintaining coordination between backup protection zone and primary protection zone relays. Coordination is maintained when relays in the primary protection zone operate faster than relays in the backup protection zone with a time difference which is equal to or more than the set coordination time interval.

Therefore, an objective function which minimises the operating time of the main relays was used. The coordination time interval requirement was used as the main constraint to the optimization problem.

It was observed that conventional overcurrent relay coordination method is able to provide settings which maintain selectivity or coordination requirements for the 230kV network of the IEEE 24 bus system. The three Evolutionary Algorithms were also able to provide settings for the directional overcurrent relays which maintain selectivity or coordination.

BGA provided parameter settings which are the same as the settings provided by the conventional relay coordination method for 1500 generations. The parameter settings for directional overcurrent relays determined by the conventional relay coordination method results in the total operating time of 18.65 seconds for main relays. The parameter settings for directional overcurrent relays determined by BGA relay coordination method results in the total operating time of 18.62 seconds for main relays. GA provided parameter settings for directional overcurrent relays that are slightly higher for 1500 generations. This parameter settings provided by GA relay coordination method results in the total operating time of 19.84 seconds for main relays. PBIL relay coordination method produced parameter settings for directional overcurrent relays which are higher than all the other methods including conventional relay coordination method. This parameter settings of PBIL relay coordination method results in the total operating time of 48.02 seconds for main relays.

The fitness value for the BGA algorithm drops rapidly in the beginning of the search from the initial value of 718 seconds to 70 seconds where it begins to settle to a fitness value in 7 iterations. It settles between 70 and 65 seconds for about 28 iterations. The algorithm maintains diversity and converges steadily until it reaches a fitness value that is within 5%

of the final fitness value (19.55 seconds) in 1008 iterations. It reaches the final value of 18.62 seconds in 1132 iterations.

The GA algorithm starts at the fitness value of 703 seconds and drops to 64 seconds in 49 iterations. It settles to 64 seconds for a few iterations and gradual converges until it reaches the fitness value that is within 5% of the final fitness value (20.80 seconds) in 1119 iterations. It reaches the final fitness value of 19.84 seconds in 1456 iterations.

The PBIL algorithm starts at the fitness value of 814 seconds and reaches a fitness value of 74 seconds in 77 iterations. The algorithm reaches the fitness value that is within 5% of the final fitness value after 397 iterations and slowly converges until it reaches the final fitness value of 48.02 seconds.

These results show that in the beginning of the search, BGA converges much faster than both GA and PBIL. PBIL explores the search spaces much more than GA and BGA in the beginning of the search. The reason for this is that unlike both the GA and PBIL, BGA uses only a portion of the best solution in the evolution process.

Furthermore, these results show that in the later stages of the search, in contrast to BGA and GA, PBIL converges prematurely; hence it yields highest fitness value compared to BGA and GA algorithms.

The simulation results also showed that the learning rate of PBIL has an impact on the performance of the algorithm in terms of the convergence rate. The learning rate did not improve the performance of the algorithm to be better than BGA and GA.

For the fixed learning rate scheme, at learning rate LR = 0.01 the algorithm explore the search space much more in the beginning and fails to converge after 600 generations. The learning rates LR = (0.02 – 0.04), not included in the report, exhibit the same behaviour. At LR = 0.05, the algorithm reach to within 50% of the final fitness value in 120 iterations, within 20% in 177 iteration, within 5% in 441 iterations and reaches the final fitness value of 66.79 seconds in 597 iterations . At LR = 0.2the algorithm reach to within 50% of the final fitness value in 50 iterations, within 20% in 115 iterations, within 5% in 387 iterations and reaches the final fitness value of 46.78 seconds in 600 iterations. At LR = 0.4the algorithm reach to within 50% of the final fitness value in 34 iterations, within 20% in 83 iterations, within 5% in 331 iterations and reaches the final fitness value of 43.81 seconds in 595 iterations. At LR = 0.6the algorithm reach to within 50% of the final fitness

value in 27 iterations, within 20% in 63 iterations, within 5% in 496 iterations and reaches the final fitness value of 45.67 seconds in 600 iterations. At LR = 0.8the algorithm reach to within 50% of the final fitness value in 27 iterations, within 20% in 53 iterations, within 5% in 366 iterations and reaches the final fitness value of 49.03 seconds in 598 iterations. At LR = 1.0the algorithm reach to within 50% of the final fitness value in 32 iterations, within 20% in 47 iterations, within 5% in 104 iterations and reaches the final fitness value of 49.24 seconds in 593 iterations.

These results show that at very low values of learning rate LR = 0.01 to LR = 0.04, the PBIL algorithm explores the search space more and lacks the ability to exploit the search space. As a result, the algorithm does not converge at these learning rate values. The algorithm starts to converge at the learning rate of LR = 0.05. At this learning rate it converges to a higher value compared to when the higher values of learning rate are used. The results further show that as the learning rate increases the faster the algorithm converges to a fitness value. This is due to the ability of the algorithm to exploit the search space much more at higher learning rates. However, the possibility of the algorithm leading to premature convergence increases at higher learning rates. This can be seen by the higher fitness values at learning rate values of LR = 0.6 to LR = 1.0. The adaptive learning rate scheme exhibit similar results.

Compared to both the GA and BGA, these learning rate schemes for PBIL are not able to improve the performance of the algorithm to be better than GA and BGA.

Table of Contents

List of Figures

List of Tables

Nomenclature

IDMTL	Inverse Definite Minimum Time Lag
t	Operating time of the relay
TMS	Time multiplier setting
$K, \propto$	IDMTL constants
I_F	Measurement fault
I_{PU}	Overcurrent pickup current setting
t_m	Operating time of main relays
t_b	Operating time of backup relays
CTI	Coordination Time Interval
Δt	Grading margin (with relation to the CTI)
J	Objective function
μ	Crossover point
β	Uniform random number between 0 and 1
LR	Learning Rate
ff	Forgetting factor
PV	Probability Vector
ρ_n	Penalty Function

CHAPTER 1

1 INTRODUCTION

This chapter introduces the concept of relay coordination with emphasis on directional overcurrent relays. Various optimization techniques applied to ensure relay coordination for directional overcurrent relays are highlighted with emphasis on Evolutionary Algorithms (GA, BGA and PBIL) which is the subject of this thesis.

Power systems are designed to be as fault-free as possible through careful design, proper equipment installation and periodic equipment maintenance [1]. However, even when these practices are followed, it is not practical to design a power system so as to completely eliminate faults from occurring [1], [2]. A fault is defined as any abnormal condition which causes a reduction in the insulation level of primary plant equipment (phase conductors, or between phase conductors and earth) and consequently results in an excess current and a drop in voltage due to a reduction in the impedance between conductors or between conductors and earth [3].

Faults are caused by the following factors amongst others: weather; equipment failure; forestry contact; public contact; animal contact; vandalism and vehicle accidents [2], [3]. The main effects of faults on the power system are: damage to the faulted plant; damage to the rest of the power system; interruption of the supply of energy to consumers; safety hazards to general public and utility personnel [2],[3].

Therefore, it is very important to eliminate or minimize the impact of the faults. To this end, power system protection equipment is used to detect abnormal conditions on the power system and swiftly isolate these conditions from the system to protect the power system from the adverse effects of abnormal conditions. In order for the protection system to perform its function optimally, it must, amongst other important characteristics, operate as fast as possible while it is able to disconnect only the faulted part of the power system leaving the rest of the healthy system undisturbed [3]. These two requirements ensure that the effects of a fault on the power system are minimized by isolating the fault as soon as possible. In addition, they ensure that the fault does not result in a disconnection of a larger part of the power system. To achieve this objective, relay coordination study must be carried out [4].

1.1 Background to the investigation

Coordination of protection is defined as the process of choosing settings or time delay characteristics of protective devices, such that the operation of the devices will occur in a specified order to minimize customer service interruption and power system isolation due to a power system disturbance [5].

Inverse Definite Minimum Time (IDMT) over-current relays are the most widely used protection system relays to detect and isolate faults on the power system. This type of relays is often used as main protection or backup protection in radial and interconnected power system networks. For IDMT overcurrent relays, the task of performing relay coordination involves the setting of pick up current and time multiplier parameters. The most important parameter for overcurrent relay coordination is the time multiplier which has a direct influence on the operating time of an overcurrent relay. An increase in the time multiplier results in an increase in the operating time of the overcurrent relay and vice versa. The pick-up current is set to be stable for maximum load current the equipment can carry continuously but should also be sensitive enough to detect minimum fault at the end of intended reach [6].

For many years, power system engineers have relied on conventional methods to provide overcurrent relay coordination. A typical conventional relay coordination method is discussed in detail in Chapter 2. These methods are based on an iterative trial and error process and can be laborious and time consuming depending on the complexity of the power system network to be coordinated. As a result, in the 1960's, it has been proposed to automate the relay coordination procedure using computer programs [7], [8]. However, these automated methods are reported to be computationally intensive and do not provide optimal solutions but rather the best of the tried solutions [9].

A comprehensive review of different optimal coordination methods to overcome this problem is given in [10], [11]. From these reviews and independent literature review conducted for this research, the following evolutionary algorithms have been applied for optimal relay coordination:

- Genetic Algorithm [12] – [16]
- Evolutionary Programming [17], [18]

- Differential Evolution [19] – [22]

Relay coordination using conventional optimization methods initially proposed has drawbacks There is a limitation in terms of number of constraints the conventional coordination algorithms can handle [10]. Moreover, these algorithms cannot take all systems conditions into consideration and therefore results obtained are always trapped in local optimum relay settings [17].Evolutionary algorithm optimization techniques (GA, EP and DE) are proposed to deal with the setbacks of conventional optimization methods. The first researchers to apply evolutionary algorithms techniques to the problem of relay coordination are So *et al* in 1997 [12]. Since then many evolutionary algorithms have been applied by other researchers as indicated above. Researchers have reported improvement when evolutionary algorithms are compared with conventional optimization techniques.

As it can be seen from the preceding discussions, not all available evolutionary algorithms have been investigated. The objective of this research is to extend the previous work and include new Evolutionary Algorithms, specifically, Breeder Genetic Algorithm (BGA) and Population Based Incremental Learning (PBIL) algorithms. Genetic Algorithm (GA) has been applied extensively on the problem of relay coordination previously with researchers reporting satisfactory results. Therefore, a comparative study of these new evolutionary algorithms with the GA for directional overcurrent relay coordination problem is useful given the success and ease of use of these algorithms in other engineering problems such as small signal stability. The University of Cape Town has done research in the past on the application of these new evolutionary algorithms on the problem of small signal stability and satisfactory results have been reported [23], [24].

1.2 Evolutionary Algorithms to be compared

Three evolutionary algorithms were selected for comparison, namely, GA, BGA and PBIL.

Genetic Algorithm

Genetic Algorithm (GA) is a random adaptive search algorithm based on the mechanisms of natural selection and natural genetics [25], [26]. This algorithm applies the principles of survival of the fittest to search for optimal solutions [25]. Solutions from one population, initialized arbitrarily, are used to create a new population through genetic operators such as selection, crossover and mutation. Genetic Algorithm was first developed by John Holland in his research to [25]:

- Explain the adaptive processes of nature
- To design the artificial systems software that retains the natural mechanism of nature.

Breeder Genetic Algorithm

The Breeder Genetic Algorithm is similar to Genetic Algorithm; however BGA is based on artificial selection unlike GA which is based on natural selection. The artificial selection used in BGA is similar to that used for animal breeding [27]. In this research, a modified version of BGA known as Adaptive Mutation BGA (AMBA) is used [28].

Population based Incremental Learning

Population Based Incremental Learning (PBIL) combines aspects of Genetic Algorithm with Competitive Learning [28]. PBIL was first proposed by Baluja in 1994. In PBIL crossover/recombination is not applied and the role of population is redefined [28]. The probability vector is used to create new trial solutions through learning. The probability vector is initially set to 0.5 to ensure that the probability vector is unbiased and the initial trial solutions created from the probability vector are completely random [28].

1.3 Justification of research

This research is a continuation of the work previously done in analysing and comparing the application of evolutionary algorithms for optimal relay coordination. Breeder Genetic Algorithm (BGA) and Population-Based Incremental Learning (PBIL) Algorithm have been recently applied with success to other engineering problems such as small signal stability problems. It is therefore useful to investigate their application on relay coordination engineering problem.

1.4 Objectives of the report

The main objectives and contributions of this research are:

- To apply the selected evolutionary algorithms on a typical power system network and study results comparatively for the three selected Evolutionary Algorithms.
- To study the effects of learning rate (LR) parameter on the performance of PBIL for relay coordination.

1.5 Research methodology

To successfully conduct a study on the application of the three selected algorithms for optimal relay coordination, a review of overcurrent relay coordination was conducted, followed by a review of the three evolutionary algorithms. The application of the algorithms was tested on the 230kV network of the IEEE 24 bus system. The coordination of directional overcurrent relays on the system was carried out using the three algorithms and the results were compared. To apply the Evolutionary Algorithms, the problem of relay coordination was formulated as an optimization problem. The objective function which minimises the total operating time of the main relays was used. Coordination requirement was used as the main constraint. Other constraints are the lower and upper bounds of the parameters.

1.6 Limitations and scope of investigation

The scope of this research is limited to the comparison of the three Evolutionary Algorithms (GA, BGA and PBIL) for directional overcurrent relay coordination. Relay types such as distance protection relays are not considered. Only directional overcurrent relays of the IEC 255-4 standard inverse type are considered in this research. Overcurrent relay characteristics such as very inverse or definite time are not considered.

Due to lack of benchmark power systems networks for relay coordination studies, the 230kV network of the IEEE 24 bus reliability benchmark power system is used to carry out the investigation into relay coordination.

1.7 Software packages used for the Simulations

The Digsilent Powerfactory tool has been used in modelling and obtaining the fault currents for the IEEE 24 bus system. For application of Genetic Algorithm, a continuous genetic algorithm code from Haupt & Haupt in [29] was used. For application of BGA, and PBIL, the codes used in this dissertation were adapted from the ones developed by John Greene from the University of Cape Town [28].

1.8 Outline of the thesis

The thesis report is organised as follows

Chapter 1: Introduction:

This chapter provides the aim of the project, background to the project, objectives of the project and research methodology.

Chapter 2: Directional Overcurrent Relay Coordination

This chapter gives an introductory account to the theory of overcurrent relay coordination. Various methods which are used to carry out overcurrent relay coordination are also discussed.

Chapter 3: Evolutionary Algorithms Theory

In Chapter 3, the evolutionary algorithms used in this research are discussed.

Chapter 4: Application of Evolutionary Algorithms to Overcurrent Relay Coordination

This chapter presents an overview of how evolutionary algorithms are applied to overcurrent relay coordination. The problem of directional overcurrent relay coordination is formulated with the aid of previous work done in this field.

Chapter 5: Simulation Results and discussions.

In this chapter, the results and discussions for three Evolutionary Algorithms are covered. The results of the algorithms are compared using, average fitness value, convergence rate, the overall operating time of main relays, operating time of back-up relays and the grading margin time.

Chapter 6: Conclusions

The conclusions based on the results of the research are given in this chapter.

Chapter 7: Recommendations

Recommendations for future research are given in this chapter.

CHAPTER 2

2 DIRECTIONAL OVERCURRENT RELAY COORDINATION

This chapter presents an overview of relay coordination and the different overcurrent relay coordination techniques as discussed [3, 6, 30].

2.1 Introduction

Coordination of protection is classically defined as "*the process of choosing settings or time delay characteristics of protective devices, such that the operation of the devices will occur in a specified order to minimize customer service interruption and power system isolation due to a power system disturbance"* [5]. To minimize the extent of the power system disconnected during a fault, the protection system is arranged to operate in zones. Protection devices in a non-unit protection system do not have a clearly defined zone and can reach into the protection zone of adjacent devices. Therefore, these devices have to be coordinated with adjacent protective devices in order to carry out selective clearing of faults on the power system.

This is achieved by applying protective devices to a power system as primary and backup pairs. In this set up, primary protection is set to operate faster for faults in their primary protection zone while backup protection is set to operate with a predetermined time delay for faults in their backup protection zone. Overcurrent relays are the most widely applied non-unit protection system on the power system.

Section 2.2 discusses coordination by time. Coordination by current magnitude is discussed in Section 2.3. Section 2.4 discusses coordination by both current and time focusing on IDMTL standard inverse overcurrent relays.

2.2 Coordination by time

In the coordination by time method, the adjacent protection devices which form primary and back-up protection pairs are arranged to operate in graded times. The protection device closest to the fault is arranged to trip in the shortest time, each adjacent breaker back to the source operates in longer times after a predetermined time delay. For a simple radial power system in Figure 2.1, an overcurrent relay at A would be set to operate faster for a fault

close to A than a relay at B and C for a fault close to A. An overcurrent relay at B is set to operate faster than an overcurrent relay at C for a fault at B, and so on.

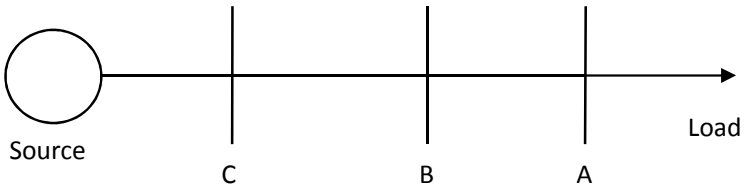

Figure 2.1 A simple radial power system network

The predetermined time delay between the operating times of the adjacent relays is referred to as grading margin. The grading margin for the protection system is used to accommodate the following factors [3, 6, 30]:

- Relay timing errors
- Allowance for CT ratio errors
- Circuit breaker interrupting time
- Relay overshoot time
- Safety margin

For earlier electromechanical relay technology the typical grading margin of 0.4s is used due to the longer overshoot time and safety margin, whereas a grading margin of 0.3seconds is used for newer numerical relay technology. In this research, a grading margin of 0.3 seconds is used.

2.3 Coordination by current magnitude

Due to differing impedance between source and the location of faults for different fault locations, the fault currents produced by faults in different locations of the power system will differ in magnitude. Therefore, current magnitudes can be used to arrange for the relay nearer to the fault to operate first before upstream relays. The relay at C in Figure 2.1 can be arranged to trip line BC for a fault at B and relay at B can be arranged to trip line BA for a fault at A, and so on. Relays that use this method of coordination are known as instantaneous overcurrent relays.

2.4 Coordination by both current magnitude and time

The main disadvantage of coordination by time only is that protection devices closer to the source where fault levels are higher will operate in longer time delays. On the other hand, discrimination by current only has a disadvantage in the fact that in cases where the impedance between two fault locations on the power system does not change significantly, the magnitude of the fault current will also not change. It will therefore be difficult for the protection devices to distinguish between the in-zone fault and the out of zone fault. To address these limitations, coordination by both current magnitude and time has been introduced.

In this coordination method, protective devices applied at locations A, B and C in Figure 2.1 are designed to use inverse operating characteristics- the higher the fault current, the faster the protective device operating time and the lower the fault current the slower the device operating time. For example, for a fault close to breaker A, the protective device at A will be typically set to operate in 0.2 seconds, but will operate in 0.8 seconds for a fault near breaker B. For a fault near breaker B, the protection device at B will operate faster than the protection device at A. In this way, coordination and speed of operation are improved.

Protection relays that use inverse operating characteristics are known as Inverse Definite Minimum Time Lag (IDMTL) relays. The IEC 255 standard mathematical formula for modelling the operating characteristics of IDMTL relays is indicated in the Equation 2.1:

$$t = TMS \times \left(\frac{K}{\left(\left(\frac{I_F}{I_{PU}} \right)^{\propto} - 1 \right)} \right) \tag{2.1}$$

I_{PU} = relay current setting and I_F = measured fault current

TMS = relay time multiplier setting.

K and α are constants for different relay characteristics as shown in Table 2.1.

Table 2.1 Relay constants for various IEC 255 IDMTL relays

Relay Characteristic	α	K
Normal Inverse	0.02	0.14
Very Inverse	1.0	13.5
Extremely Inverse	2.0	80
Long-time Inverse	1.0	120

In an interconnected network in which current can flow in either direction of the overcurrent relay, overcurrent relays are configured to operate in one direction only. This type of relays is known as the directional overcurrent relay. Directional overcurrent relays are arranged to operate when current is flowing into the line [30]. Coordination of directional IDMT overcurrent relays involves selection of relay pick-up current setting, I_{PU} and time multiplier setting, *TMS*.

The IDMT relay parameters have the following effects on the IDMT relay curves [31]:

Adjusting the relay pick-up setting moves the curve left or right in the horizontal direction as shown in Figure 2.2.

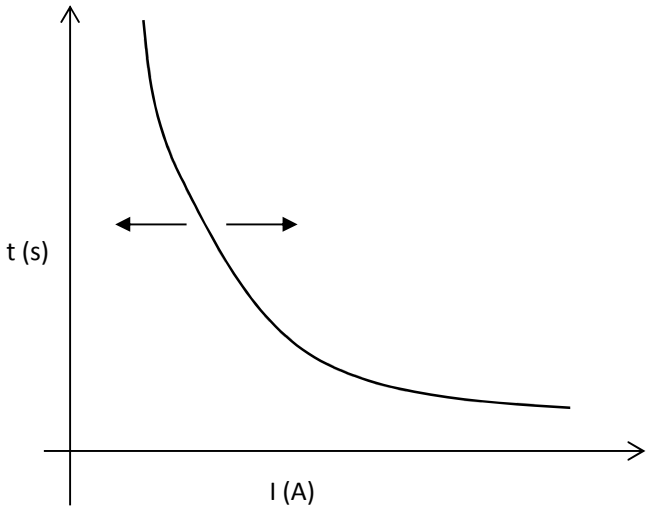

Figure 2.2 The effect of current pickup setting on the relay operating curve [31]

Adjusting the time multiplier setting moves the curve up and down in the vertical direction as shown in Figure 2.3.

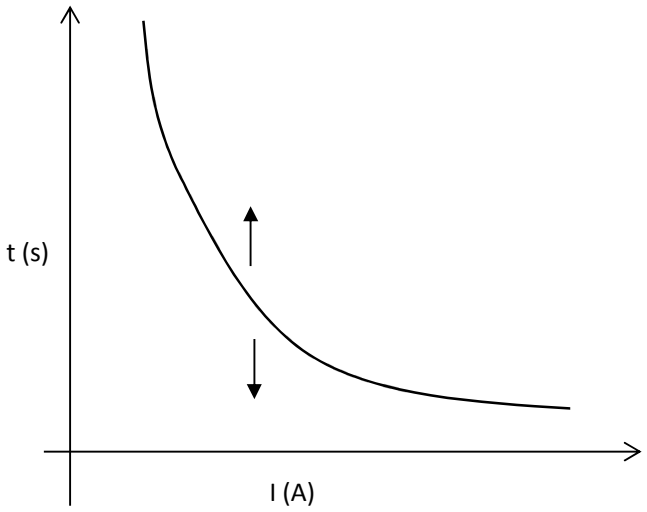

Figure 2.3 The effect of time multiplier setting on the relay operating curve [31]

In a properly coordinated protection system, the IDMT relay curves must not cross at minimum and maximum fault current, as indicated in the Figure 2.4.

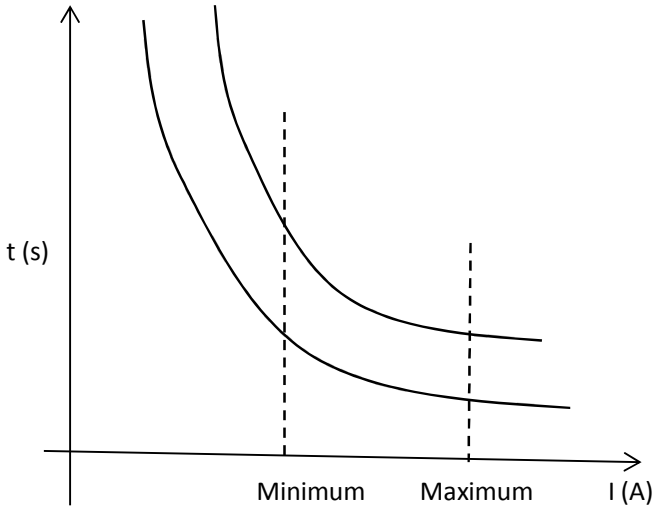

Figure 2.4 An example of relay curves with proper coordination [31]

2.4.1 Guidelines for selecting relay I_{PU} and TMS parameters

Guidelines for selecting the parameters I_{PU} and TMS parameters for directional IDMT overcurrent relays to ensure proper relay coordination are as follows [6]:

2.4.1.1 Determine the relay minimum operating current

This current is also known as the relay pick-up current setting and it defines the pick-up current of the relays. For phase overcurrent protection, the pick-up setting is determined by allowing a margin of overload above the protected equipment nominal current rating. The following criteria are suggested [6]:

- The pickup setting current must be set to 50% or less than the minimum end-of line phase to phase fault.
- The setting must be checked to be 200% or more than the maximum load current.
- Determine the relay critical fault current

Relays are coordinated in pairs with the upstream relay set to coordinate properly with the downstream relay. The current at which the relays are coordinated is known as the critical current. For inverse relays this is always almost the maximum current at the downstream relay.

2.4.1.2 Determine the relay time dial

The time dial setting adjusts the time delay before the relay operates whenever the fault current reaches the value equal to or greater than the relay current setting. In order to obtain appropriate protection and coordination, the following procedure is used to select the time dial setting or time multiplier setting [6]:

Step 1: Determine the operating timet_1, of the relay furthest away from the source for a critical fault current just in front of its associated breaker.

Step 2: Determine the operating time of the next upstream relay towards the source,

$t_{2a} = t_1 + CTI$.Use the same fault level or critical fault current used to determine t_1.

Step 3: Determine the operating time,t_{2b} of relay in the previous step for a fault just in front of its associated breaker.

Continue with the sequence for the next relay upstream towards the source, starting from second stage until the parameters for all relays are selected.

This process works perfectly for a radial system. However, coordinating a loop system is much more complex due to different fault currents measured by relays in a coordination pair for a fault [30]. In a looped system, directional overcurrent relays are coordinated around the loop according to their direction- clockwise or anticlockwise.

A more refined process for coordination for a loop system is defined with and aid of an example in [30]. This process can be summarised as follows:

Step 1: Determine relays to be coordinated around the loop (in either direction)

Step 2: Initialize all relays to operate in 0.2s for close-in faults

Step 3: Choose an arbitrary relay as a starting point

Step 4: Determine the relays which take part in the coordination pair with the chosen relay

Step 5: Set the TM of the starting relay to operate in (main operating time of the main relays + CTI) for the far end fault.

Step 6: Proceed to the next relay in the loop.

Step 7: Perform Steps 4 – 5

Step 8: Perform a coordination check between the relay and the previous relay in the loop. If the coordination requirement is met, repeat Step 6-7. If the requirement is not met adjust the TM setting (determined in Step 5) of the previous relay to meet the criteria.

The process is repeated to the next relay around the loop until all the relays in the network are coordinated.

2.5 Summary

In this chapter, various overcurrent relay coordination techniques were discussed:

- Coordination by time
- Coordination by current magnitude
- Coordination by current and time.

The chapter was concluded by providing guidelines on how to select the current pickup parameter and the time multiplier setting. In the next chapter, an overview of evolutionary algorithms will be discussed focusing on the three selected algorithms to be compared.

CHAPTER 3

3 EVOLUTIONARY ALGORITHMS THEORY

This chapter presents an overview of evolutionary algorithms and different evolutionary algorithms are discussed.

3.1 Introduction

Evolutionary Algorithms (EAs) are a family of population based stochastic algorithms that simulate the process of natural evolution. They are based on the concept of Darwinian evolution of *"survival of the fittest"* which is characterized by the natural processes of reproduction, mutation, competition and selection [26], [32].

This chapter presents an overview of evolutionary algorithms and the different evolutionary algorithms techniques. Section 3.2 gives an overview of evolutionary algorithms; the following topics common to the evolutionary algorithms are also discussed in this section: fitness function and selection schemes. An overview of Genetic Algorithm is discussed in Section 3.3. Section 3.4 discusses an overview on Breeder Genetic Algorithm. An overview of Population Based Incremental Learning is discussed in Section 3.5.

3.2 Overview of Evolutionary Algorithm

The following overview of how the Evolutionary Algorithms work is based on [26], [32]. In solving an optimization problem, these algorithms work on the concept of a population of individuals which represents potential solutions to a given optimization problem. The population is initialized arbitrarily to ensure that it is a uniform representation of the search space. These populations evolve towards better regions of the search space based on each individual's fitness through the random process of selection, mutation and reproduction operators. The fitness or the quality of an individual is quantified by the evaluation of the fitness function. The fitness function is formulated in such a way that it represents the given optimization problem. The selection operator leads the algorithm towards better solutions by favouring individuals with higher fitness to reproduce new solutions more often than individuals with lower fitness. New solutions ("offspring") are created from selected individuals ("parents") through reproduction operators, namely, crossover/recombination and mutation operators. Crossover operator creates new solutions

by combination of information from two parents. On the other hand, mutation introduces new information to the search space by randomly varying individuals in the population.

3.2.1 Fitness Function

Each individual in the population represent a potential solution to the given problem. To measure how close a solution is to the optimal solution, the fitness function is evaluated. The fitness function used in this thesis will be common to all the EAs selected for analysis and comparison. Details on the formulation of the objective function are discussed in Chapter 4.

3.2.2 Selection schemes

The selection methods that are frequently used in Evolutionary Algorithms are summarised below. More details on the different selection methods can be found in [26], [29]:

- **Random Selection**

In this selection method individuals in the population are selected arbitrarily without assessing their fitness.

- **Proportional Selection**

In this selection method the chance of the individual being selected is determined according to the individual's relative fitness. The probability of the individual being selected to participate in the reproduction of next population is directly proportional to the individual's relative fitness.

- **Tournament Selection**

In the tournament selection method two individuals are chosen randomly form the population. These chosen individuals compete against each other with the individual with the best fitness being selected.

- **Rank-Based Selection**

In this method the probability of the individual to be selected is not based on the individual's fitness but on the rank of the individual.

- **Truncation**

In this method a certain threshold (percentage) of the best individuals in terms of fitness is selected to take part in reproduction.

Reproduction techniques will be discussed in the subsequent sections. Various Evolutionary Algorithms differ based on how the following factors are modelled in a specific evolutionary algorithm:

- Encoding or representation of the solution
- Initialization
- Selection techniques
- Reproduction methods

This thesis applies three types of evolutionary algorithms, namely, Genetic Algorithm (GA), Breeder Genetic Algorithm (BGA) and Population Based Incremental Learning (PBIL) for optimal directional overcurrent relay coordination problem. The details of these algorithms are given in subsequent sections.

3.3 Overview of Genetic Algorithm

The following overview of Genetic Algorithm is based on [25], [26], [29], [33]. Genetic Algorithm (GA) is a random adaptive search algorithm based on the mechanisms of natural selection and natural genetics [25], [26]. This algorithm applies the principles of survival of the fittest to search for optimal solutions [25]. Solutions from one population, initialized arbitrarily, are used to create a new population through genetic operators such as selection, crossover and mutation. The first study on this algorithm was conducted by John Holland in his research to [25]:

- Explain the adaptive processes of nature
- To design the artificial systems software that retains the natural mechanism of nature.

This algorithm is different from traditional optimization system in the following aspects [25]:

- They work with coding of the parameter, not the parameter itself

- They search for a population of points (parallel search), not a single point
- GA use objective function for optimization not the derivative or other auxiliary knowledge
- They use probabilistic rules not deterministic rules

3.3.1 Solution Encoding or representation

The most common representation scheme for GA is binary vector of fixed length. Other parameter representation schemes such as Gray coding and real-valued representation can be used [26]. In this thesis, the Genetic Algorithm used from [29] uses real-valued or continuous representation.

3.3.2 Selection

In this thesis, the Genetic Algorithm uses the rank-based selection method. In this method, a probability of selecting an individual solution is assigned based on the rank of the individual solution when all the solutions are sorted according to the fitness. The fitness of the individual is determined by evaluating the value of the individual according to the objective function. This method has an advantage in that the fittest individual will not dominate in the selection since the probability is based on ranking not only the fitness. In [29], the method is implemented as follows:

The probability of the individual in the population is determined according to this Equation (3.1):

$$P_n = \frac{N_{keep} - n + 1}{\sum_{n=1}^{N_{keep}} n} \tag{3.1}$$

Where n is the individual in the population and N_{keep} is the number of individuals in the population selected to take part in reproduction of new population.

A random number between zero and one is generated. The first number with the cumulative probability is selected to participate in the reproduction of new population of potential solutions.

3.3.3 Recombination or Crossover

A new population is created by combining the information of the parent solutions selected in the selection process. The aim of crossover function is to produce new solutions from

parent solutions in order for the algorithm to search through the solution space [25], [26]. Crossover is done through various methods such as [26], [29]:

- Simple crossover,
- Arithmetic crossover
- Heuristic crossover.

In this research, a combination of extrapolation technique and arithmetic crossover for Genetic Algorithm is applied as follows [29]:

A variable in the first pair of parents is selected randomly to be the crossover point.

$$\mu = roundup(random * N_{var}) \tag{3.2}$$

Supposing two parents selected for reproduction to be $parent_1$ and $parent_2$ as below

$$parent_1 = [p_{m1} p_{m2} \dots p_{m\mu} \dots p_{mNvar}] \tag{3.3}$$

$$parent_2 = [p_{d1}\, p_{d2} \dots\ p_{d\mu} \dots p_{dNvar}] \tag{3.4}$$

The selected crossover point for the two parents pair is $p_{m\mu}$ and $p_{d\mu}$for $parent_1$ and $parent_2$ respectively.

New variables that will be inserted in the offspring at the crossover point are then formed by combining the selected crossover point from each parent as below

$$p_{new1} = p_{m\mu} - \beta[p_{m\mu} - p_{d\mu}] \tag{3.5}$$

$$p_{new2} = p_{d\mu} + \beta[p_{m\mu} - p_{d\mu}] \tag{3.6}$$

Where β is a random number between 0 and 1. Crossover is completed by swapping the variables of the pair as before.

$$offspring_1 = [p_{m1} p_{m2} \dots p_{new1} \dots p_{dNvar}] \tag{3.7}$$

$$offspring_2 = [p_{d1}\, p_{d2} \dots p_{new2} \dots p_{mNvar}] \tag{3.8}$$

If the first variable of the individual is selected then only the variables to the right of the crossover point are swapped. If the last variable of the individual is selected then only the variables to the left of the crossover point are swapped.

3.3.4 Mutation

To prevent premature convergence of the algorithm, diversity is introduced to the population to broaden the search capability of the algorithm by the mutation operator. There are various methods to implement mutation such as:

- Uniform mutation,
- Non-uniform mutation
- Multi-non-uniform mutation.

In this research, uniform mutation is implemented [29]. In uniform mutation a randomly selected variable is set to equal to a uniform random number with in the specified range [29].

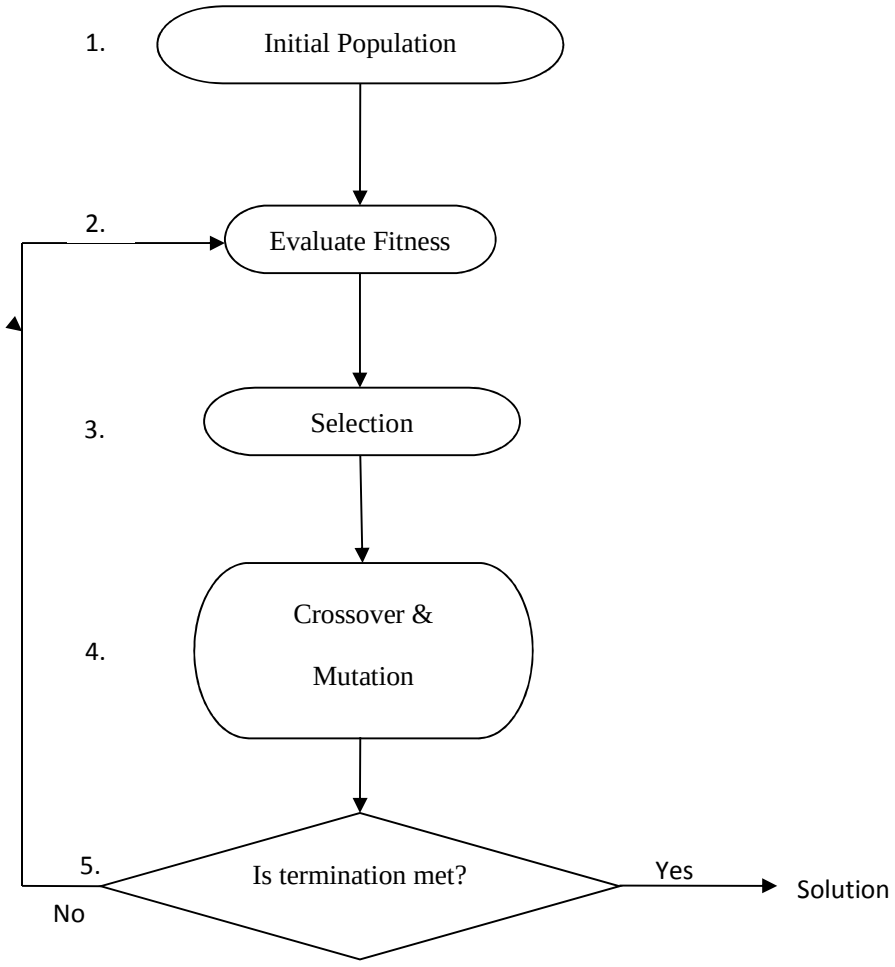

Figure 3.1 The flow chart of a simple GA and BGA

A summary of the GA, based on the Continuous Genetic Algorithm Optimization from [29], used in the research work is given below:

Step 1: Create an initial population (usually a randomly generated string)

Step 2: Evaluate all of the individuals (apply some function or formula to the individuals)

Step 3: Select a new population from the old population based on the fitness of the individuals as given by the evaluation function.

Step 4: Apply some genetic operators (mutation & crossover) to members of the population to create new solutions.

Step 5: Check for termination criterion. If not met, Repeat Step 2- Evaluate these newly created individuals, until criteria is met. If the criterion is met, take the best solution as the solution to the given problem.

3.4 Overview of Breeder Genetic Algorithm (BGA)

The Breeder Genetic Algorithm is similar to Genetic Algorithm; however BGA is based on artificial selection unlike GA which is based on natural selection. In natural selection, selection of the population that takes part in the next generation is done in a probabilistic manner. Populations with best fitness are assigned a higher probability of taking part in the next generation than populations with lower fitness. In contrast, in artificial selection, the selection of the population is done in a deterministic manner in which a certain percentage of the best individuals are selected to take part in the next generation. The artificial selection used in BGA is similar to that used for animal breeding [27]. In this research, a modified version of BGA known as Adaptive Mutation BGA (AMBA) explained in [28] is used.

BGA is similar to GA in operation. Therefore, the flow chart of a simple GA in Figure 3.1 and the summary of GA also apply to BGA. As discussed above, the two algorithms differ mainly in the selection technique used to determine the population that takes part in the next generation. In this research, the two algorithms also differ in the type of crossover and mutation operators applied. These differences are indicated in Figure 4.2 and Figure 4.3.

3.4.1 Solution Encoding or representation

In this thesis, the Breeder Genetic Algorithm used from [28] uses real-valued or continuous representation.

3.4.2 Selection

The BGA uses truncation selection method. To perform this method, trial solutions are evaluated and ranked according to their fitness. The top T% of the best individuals of the trial solutions is selected and goes through recombination and mutation to create the next generation. The rest of the individuals are discarded. The next generation is made up of the best trial solution which is directly inserted into the next generation and the rest of the solutions are created by recombination and mutation.

3.4.3 Recombination/Crossover

There are various possible recombination methods for producing a new child solution from two parent solutions that can be implemented in Adaptive Mutation Breeder Genetic Algorithm (AMBA) [28]. Different recombination methods search the space with different bias; therefore it is advisable to include several recombination methods since it is not known in advance which bias will be suitable for a specific problem. In this research, volume crossover and line crossover methods are used [28].

In volume crossover, a random vector r equal in length to the parents' length is generated. The child z_i is produced by the following equation [28].

$$z_i = r_i x_i + (1 - r_i) y_i. \tag{3.9}$$

Where x_i and y_i are two parents.

It can be said that the child produced lies at a random point inside the hypercube defined by the parents. [28].

In line crossover, single uniformly random between 0 and 1 is generated. The child z_i is produced by the following equation [28].

$$z_i = r x_i + (1 - r_i) y_i \tag{3.10}$$

It can be said that the child produces is located a random point on a line connecting two parents, $x_i$and y_i. [28].

3.4.4 Adaptive Mutation

Since BGA uses only a certain percentage of fittest solutions as defined by the selection method, there is a possibility of premature convergence. That is, the algorithm might

converge to local optimum instead of global optimum. Similar to GA, to solve the problem of premature convergence, diversity is preserved by mutation. Mutation is implemented by adding a small vector of normally distributed zero mean random numbers to the child produced by recombination before inserting it into the next generation [28]. The magnitude of the standard deviation, rof the mutation vector is very important. A small value of r might result in premature convergence and a big value might disrupt the search and reduce the ability of the algorithm to converge. [28]. Therefore, it is advised to use adaptive approach where the mutation rate is modified during the course of the search. This is achieved by dividing the population in two halves X and Y. Initially the rate is set to nominal rate r_{nom}. A mutation rate of $2r_{nom}$ is applied to X while the mutation rate of $\frac{1}{2}r_{nom}$ is applied to Y. The nominal mutation rate is adjusted depending on the population that produces better and fitter solutions on average. If population X is producing solutions that are fitter than solutions produced by population Y, then the mutation rate is increased by say 10%. If solutions produced by population Y are fitter then the mutation is decreased by the same percentage.

3.5 Overview of Population based incremental learning

Population Based Incremental Learning (PBIL) combines aspects of Genetic Algorithm with Competitive Learning [34]. PBIL was first proposed by Baluja in 1994. In PBIL crossover/recombination is not applied and the role of population is redefined [34].The probability vector is used to create new trial solutions through learning. The probability vector is initially set to 0.5 to ensure that the probability vector is unbiased and the initial trial solutions created from the probability vector are completely random [34]. PBIL is evolutionary in a sense that the solution is improved within the algorithm independent of external factors.

3.5.1 Solution Encoding or representation

In this thesis the representation scheme used for PBIL is a binary vector of fixed length.
The manner in which the population is created is explained in Section 3.5.3

3.5.2 Competitive Learning

The ability to learn is an essential trait of intelligence. Competitive Learning is often studied in the context of Artificial Neural Networks (ANNs). A learning process in the

context of Artificial Neural Networks can be viewed as the process of updating network architecture and connection weights from available training patterns. The performance of the network is improved during training or learning by iteratively updating the weights in the network. Various learning rules which govern the weight updating process are discussed in [35]. In the competitive learning rule the outputs compete amongst themselves for activation [34], [35]. The winning output unit is allowed to fire for each point presented. This concept is often defined as winner-take-all. In a simple ANN, the activation of the output units from the inputs is calculated by Equation (3.11):

$$output_i = \sum_j w_{ij} \times input_j \tag{3.11}$$

The weights of the winning output are adjusted according to Equation (3.12):

$$\Delta w_{ij} = LR \times (input_j - w_{ij}) \tag{3.12}$$

This results in the weight vectors of the winning output being moved closer to the input vector. The next sections will discuss how this idea of competitive learning is used in the PBIL algorithm.

3.5.3 The role of Population and the Probability Vector

The PBIL creates the probability vector which is used to generate a population of solutions. The probability vector is considered to be the prototype of high evaluation vectors for the search space. Solutions are generated from the probability vector according to the Equation (3.13):

$$B(i) = random(0,1) < PV(i) \tag{3.13}$$

where $B(i)$ is the i-th element the best trial solution and $P(i)$ is the i-th element of the probability vector.

A uniformly random vector is generated and compared element-by-element with the probability vector. Wherever an element of the PV is greater than the corresponding random element, a '1' is generated, otherwise a ‘0’ in the elements of the best trial solution.

In the similar manner as the weights are updated in the competitive learning ANN, the probability vector of the PBIL is shifted towards the best solution in the generation according to the update rule in Equation (3.14):

$$PV(i+1) = (1-LR)PV(i) + LR \times B(i) \tag{3.14}$$

where LR is the learning rate.

This rule is similar to the Kohonen's competitive learning rule defined in [36], [37]. Other update rules such as using a number of best solutions to update the Probability Vector or shifting the Probability Vector away from the worst solution can be used [34]. As already mentioned, PBIL does not use the crossover operator. Therefore, the next set of solutions is then generated from the updated probability vector. In this way, the next set of solutions is made to resemble the previous best solution in order for the algorithm to converge to a globally best solution. To check whether the solution is best or not, a fitness measure using the objective function is determined. The solution that returns a high value is deemed the best. The best trial solution is then selected while the rest are ignored. This best solution is used to generate the new set of solutions until the fittest solution is found, i.e. a solution that best optimises the objective function.

3.5.4 Mutation

By employing aspects of competitive learning in which the best solution is used to update the probability vector, the PBIL algorithm has ability to converge very fast. Similar to other evolutionary algorithms, the role of mutation in the PBIL algorithm is to introduce diversity in the search space. The role of mutation is to prevent the probability vector from premature convergence [34]. In [34], a random mutation of the probability vector in which the probability vector is shifted by 0.05 in a random direction with a probability of 0.02 is proposed. In this thesis, mutation is implemented by shifting the probability vector by a small factor, known as the forgetting factor, towards a neutral value of 0.5. This is shown in the Equation (3.15)

$$PV(i+1)_{new} = ff\,(PV(i+1) - 0.5) \tag{3.15}$$

ff is the forgetting factor.

3.5.5 Learning Rate

The learning rate affects how fast the probability vector is shifted to resemble the best trial solution. Since the probability vector is used to generate new solutions, the learning rate therefore affects the algorithm's ability to explore the search space of potential solutions [34]. The setting of learning rate has a direct impact on the trade-off between the PBIL's

ability to search the search space more comprehensively and the ability of the algorithm to make use of the search that has already being done [34]. A higher learning rate results a faster convergence rate and a lower learning rate results in a more comprehensive search for the optimal solution. Therefore a learning rate that obtains balance between exploitation and exploration of the search space must be found. The effects of the learning rate on the given problem have been investigated in this research.

The flow chart in Figure 3.2 illustrates the steps taken to implement the basic PBIL algorithm for function optimization.

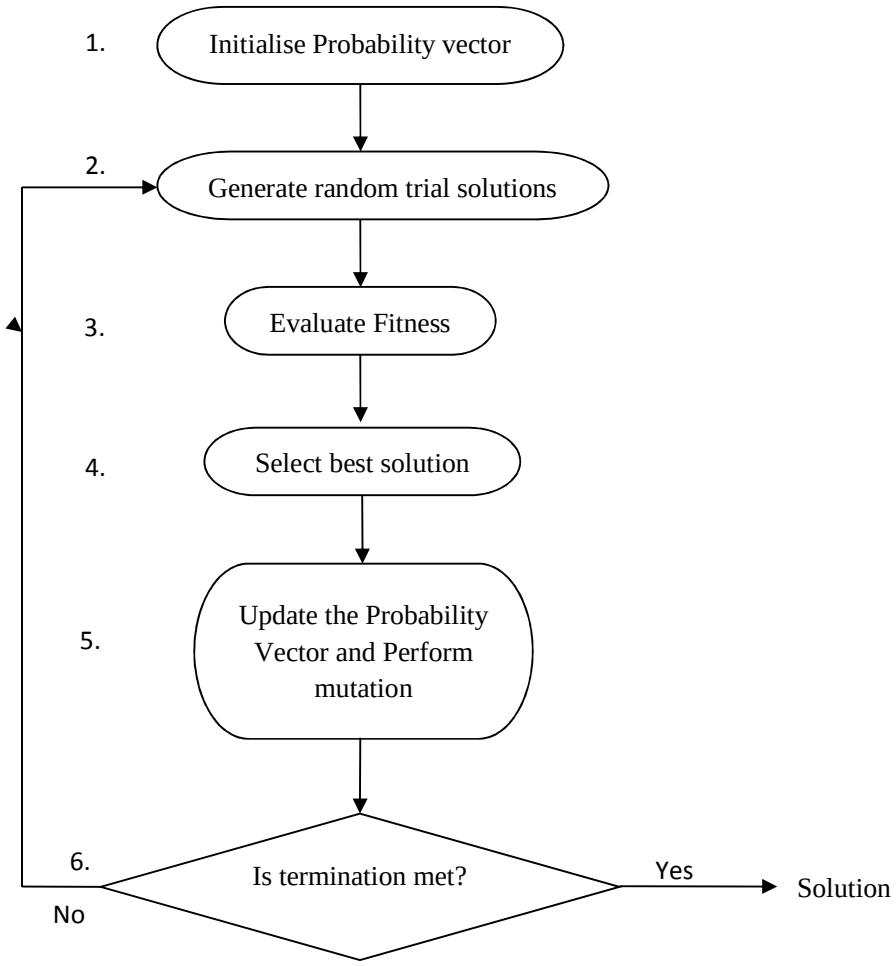

Figure 3.2 The flow chart for a PBIL algorithm

3.5.6 The basic PBIL algorithm

In step 1, the probability vector is initially set to 0.5 in order to allow for the algorithm to search for solutions that lie uniformly in the search space. The initial trial solutions are then generated randomly in step 2 according to Equation (3.13).

In step 3, the fitness of each trial solution is checked by evaluating a scalar function of these trial solutions used as a fitness measure. As is the case with all evolutionary algorithms, the fitness measure depends on the problem that has to be solved.

In Step 4, the best solution is selected. This is the solution with the highest fitness value for the given optimization problem.

The best trial solution is used in step 5 to update the probability vector such that the next generated trial solutions will resemble the previous best according to Equation 3.14. To add diversity mutation is performed in step 5. Mutation is performed according to Equation (3.15).

If the termination condition is met, in step 6, the best trial solution for all the iterations is taken as the best solution to the problem at hand. Termination condition may be met when the fitness value does not change drastically for a certain number of generations. Alternatively, termination may be met when a specified number of trial solutions generation is reached.

If termination condition is not met, the next trial solutions are generated using the update probability vector and the cycle continues until the algorithm terminates i.e. the termination condition is met.

3.5.7 Effects of Learning Rate (LR) on the Performance of PBIL

The choice of the learning rate (LR) has a big impact on the performance of the PBIL algorithm. A lower value of LR allows the algorithm to perform a more comprehensive search of the search space while higher value of LR improves the ability of the algorithm to make use of the information relating to the search space it has gathered so far [34]. As a result, for lower values of LR the algorithm takes longer to converge to an optimum value. On the other hand, for higher values of LR, the algorithm settles to a value much faster and can result in premature convergence.

To study the effects of LR the following learning rates schemes are considered, as it is done in [38]:

- Fixed LR:

  The learning rates considered are: LR = 0.01, 0.05, 0.1, 0.2, 0.4, 0.6, 0.8, and 1.0.

- Purely Adaptive LR:

  The simulation is started with a small value of LR and the value is increased linearly with the generation until the final value of LR is reached, according to Equation (3.16) [39].

$$LR(i) = \frac{G(i)}{G} \times LR_{max} \quad (3.16)$$

  where $LR(i)$ is the learning rate at the i-th generation, LR_{max} is the maximum learning rate at the end of the run,$G(i)$ is the i-th generation and Gis the total number of generations.

3.6 Summary

In this chapter, an overview of evolutionary algorithms was given. The following evolutionary algorithms were discussed:

- Genetic Algorithm
- Breeder Genetic Algorithm
- Population-Based Incremental Learning algorithm.

The flow chart for each algorithm was also given in this chapter. In the next chapter, application of evolutionary algorithms for relay coordination will be discussed. The chapter will also discuss the work previously done in this field.

CHAPTER 4

4 APPLICATION OF EVOLUTIONARY ALGORITHMS TO OVERCURRENT RELAY COORDINATION

This chapter presents an overview of how evolutionary algorithms are applied to overcurrent relay coordination. The optimization of relay coordination problem is formulated by looking at the previous work done in this field.

4.1 Introduction

The idea of solving the problem of directional overcurrent relay coordination based on principles of optimization theory was first introduced by Urdaneta *et al* [9]. The authors proposed to solve the optimization problem via linear programming optimization method. Due to limitations, such as, convergence to local optimum, in classical optimization theory used by Urdaneta *et al*, So *et al* proposed a new relay coordination method based on Genetic Algorithm [12]. Since then various researchers have proposed optimization for directional overcurrent relay coordination based on different Evolutionary Algorithms, employing a variety of approaches as far as relay operating characteristics, objective function modelling, and handling of constraints violations as well as representation of variables are concern.

The purpose of this chapter is to show how the problem of directional overcurrent relay coordination is formulated into an optimization problem. Furthermore, to illustrate how the selected Evolutionary Algorithms are applied to solve the resulting optimization problem.

Section 4.2 shows the different evolutionary methods applied previously to solve the relay coordination optimization problem. A review of different relay operating characteristic employed previously in application of evolutionary algorithms is given in Section 4.2. Section 4.3 discusses different ways the relay coordination problem can be modelled as an optimization problem. Section 4.4 discusses various options on the objective function that can be used. The issue of how constraints violations are handled is discussed in Section 4.5. The chapter ends by discussing the application of selected evolutionary algorithms to solve the given problem.

4.2 Overcurrent relay operating characteristics

Overcurrent relays can be represented by different time-current mathematical formulae. In applying evolutionary algorithms for optimal directional overcurrent relay coordination a number of mathematical models used to represent the overcurrent relay can be categorised as follows:

- IEC 255- 4 Standard
- Various operating characteristics
- Mathematical polynomial approximation

The most widely applied overcurrent relays in industry use the IEC 255-4 operating characteristics for standard inverse relays, represented by the Equation (2.1). Therefore, in this research, the IEC 255-4 operating characteristics for overcurrent relay are used.

4.3 Optimization problem modelling

TMS and I_{PU} are two parameters for the overcurrent relay that must be set for relay coordination. For any mathematical model used, these two parameters form a relationship with the operating time of the relay. For evolutionary algorithms, the relay coordination problem can be represented by the following categories of optimization problems:

- Linear problem
- Non-linear
- Mixed Integer Non-linear problem

In this research for simplicity, I_{PU} is pre-determined based on the guidelines discussed in Chapter 2 and only TMS will be optimized. Therefore, setting I_{PU}as constant Equation (2.1) becomes,

$$t = TMS \times C \tag{4.1}$$

The relationship between the relay operating time and *TMS* is thus linear; hence the optimization problem is formulated as a linear problem.

4.4 Objective function

To ensure that the evolutionary algorithms are able to solve the given problem, the objective or fitness function has to be chosen is such a way that the parameters of the directional overcurrent relays minimizes operating times for primary relays while ensuring sufficient relay coordination between primary and backup relays. Furthermore, it should be ensured that the objective function is minimized within the parameter bounds.

In general, researchers have used the operating time of main relays for primary zone fault as the objective function for the optimization problem. Other authors have incorporated additional terms, besides the constraints, in the objective function. The different ways researchers handle constraints are discussed in the subsequent section. In [12], the preferred time multiplier setting, current pick up setting and coordination time interval were added as part of the objective function. In [14], the coordination time interval is also part of the objective function. In [19], the operating time of relay for faults in the backup zone is also added as part of the objective function.

In this research, the following objective function was investigated:

- The objective function which minimizes (optimize) the operating time of the main relays:

$$f_1 = \sum t_m \tag{4.2}$$

As indicated in Chapter 2, the grading margin between main and back-up relays is very important as it allows the main relay sufficient time to trip for a primary fault before the backup relay can trip. This is done to preserve selectivity and coordination between relays.

Consequently, the optimization problem is formulated as follows:

Minimize J subject to the following constraints.

$$0.01 \leq TMS \leq 1.0 \tag{4.3}$$

$$0.5 \leq I_{PU} \leq 2 \tag{4.4}$$

$$\Delta t \geq 0 \tag{4.5}$$

J is the objective function given in Equation 4.2. The bounds of the parameters are as used in [12].

4.5 Constraints violations handling

In [12], a strategy for addressing solutions that violates any of the discussed constraints was not used. It was later shown that this method can yield overcurrent parameters that minimize the operating time of the relays but are not selective. Razavi *et al* [14] suggested improvements, which incorporated a term that compares the grading margin to a positive grading margin. If there is a mismatch the objective function is increased by a certain factor, thus penalising the infeasible solutions. In [18], constraints violations were handled by incorporating a term that looks at the number of constraints and increase the objective function value by a factor. In this research, to handle the constraints, stationary penalty functions are used [40]. That is, a penalty function penalizes the infeasible solutions by adding a penalty factor value to the objective function based on the amount of constraint violation as discussed below. The value of the penalty factor is defined by trial and error. This value must not be too big as the algorithm might not recover after being penalised. In using penalty functions, the common method is to convert all the constraints into inequality constraints of the form in Equation (4.6):

$$p - \in \leq 0 \tag{4.6}$$

where $\in$ is a small tolerance value and p is the constraint penalty function.

For relay coordination, the following penalty functions are identified based on the constraints:

$$p_1 = W_1 \sum(-\Delta t_{mb}) \tag{4.7}$$

$$p_2 = W_2 \sum(TMS - 1) \tag{4.8}$$

$$p_3 = W_3 \sum(0.05 - TMS) \tag{4.9}$$

where W_1 is used to control the weighting of "miscoordination" penalty; W_2 is used to control the weighting of upper bound penalty and W_3 is used to control the weighting of lower bound penalty.

Therefore, the following objective function, with the penalty functions incorporated has been used:

$$J = J + max(0, p_n) \tag{4.10}$$

$$p_n = \begin{cases} \sum_{n=1}^{3} -h_n \ \ if \ h_n < 0 \\ 0 \qquad\qquad otherwise \end{cases} \tag{4.11}$$

where $p_n$is the *n*-th penalty function and h_n is the *n*-th constraint function. The penalty function is added to the objective function only if there's a constraint violation.

It was shown by Razavi *et al* in [19] that treating the overcurrent parameter as continuous during the optimization process and only making them discrete at the end of the process will result in lack of coordination between relays. To address this, the authors proposed that the trial solutions from the algorithm are rounded off to the next upper available value before fitness evaluation. In this research, this proposed approach is used.

4.6 Application of Evolutionary Algorithms for Relay coordination optimization

The flow charts in Figure 4.2 to Figure 4.4 describe the procedure followed in applying the evolutionary algorithms for optimal relay coordination for the 230kV network of the IEEE24 Bus power system. The parameters that were used for the evolutionary algorithms are given in Figure 4.1.

The number of generations is set to 1500 for all the evolutionary algorithms to ensure that the evolutionary algorithms are afforded enough time to explore the search space.

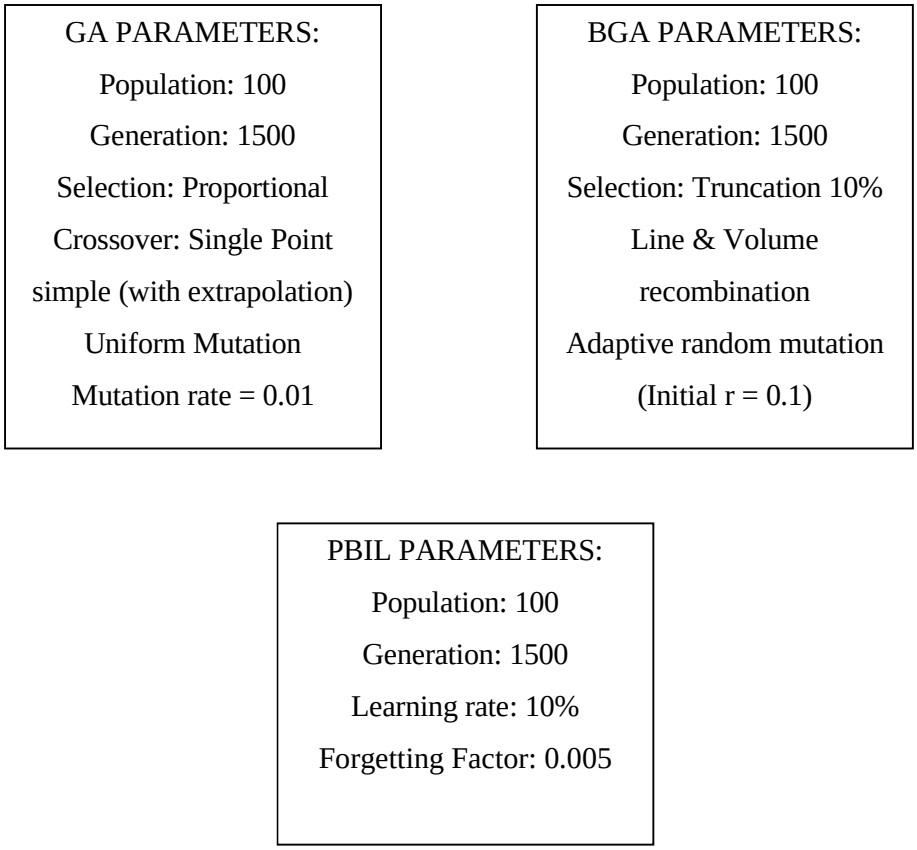

Figure 4.1The parameters of the Evolutionary Algorithms (BGA, GA and PBIL)

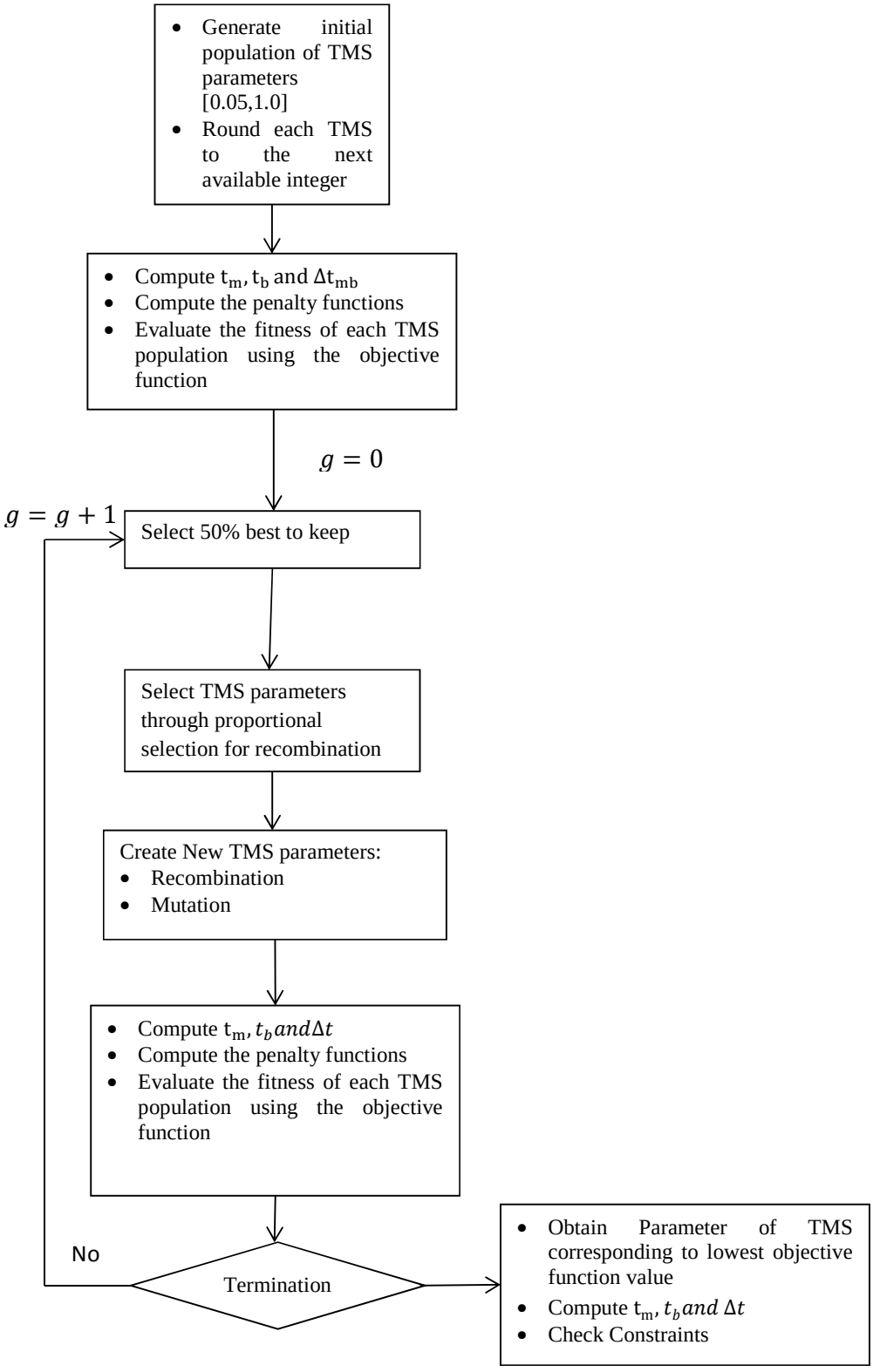

Figure 4.2 Flow chart illustrating Application of GA for relay coordination

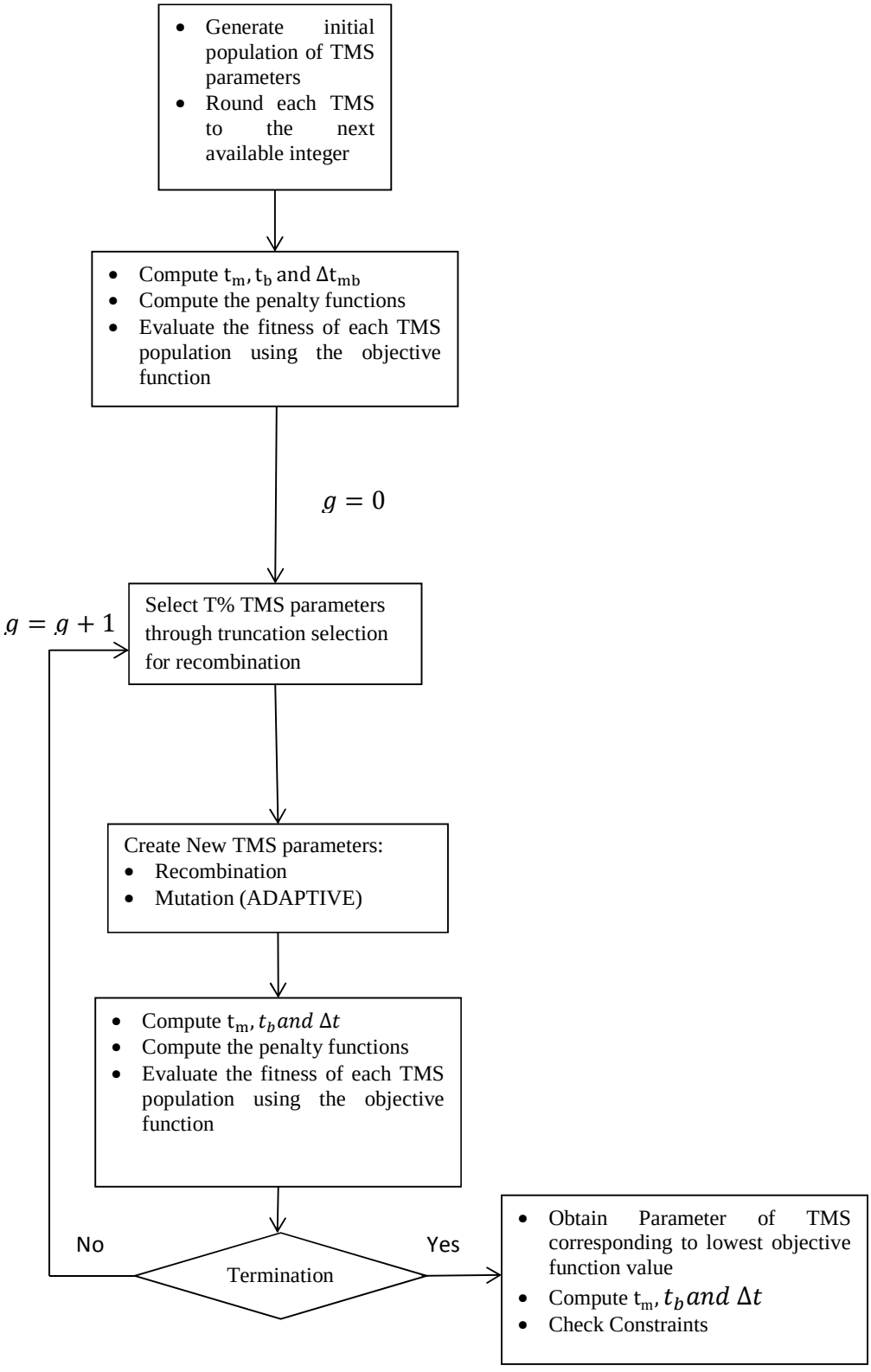

Figure 4.3 Flow chart illustrating Application of BGA for relay coordination

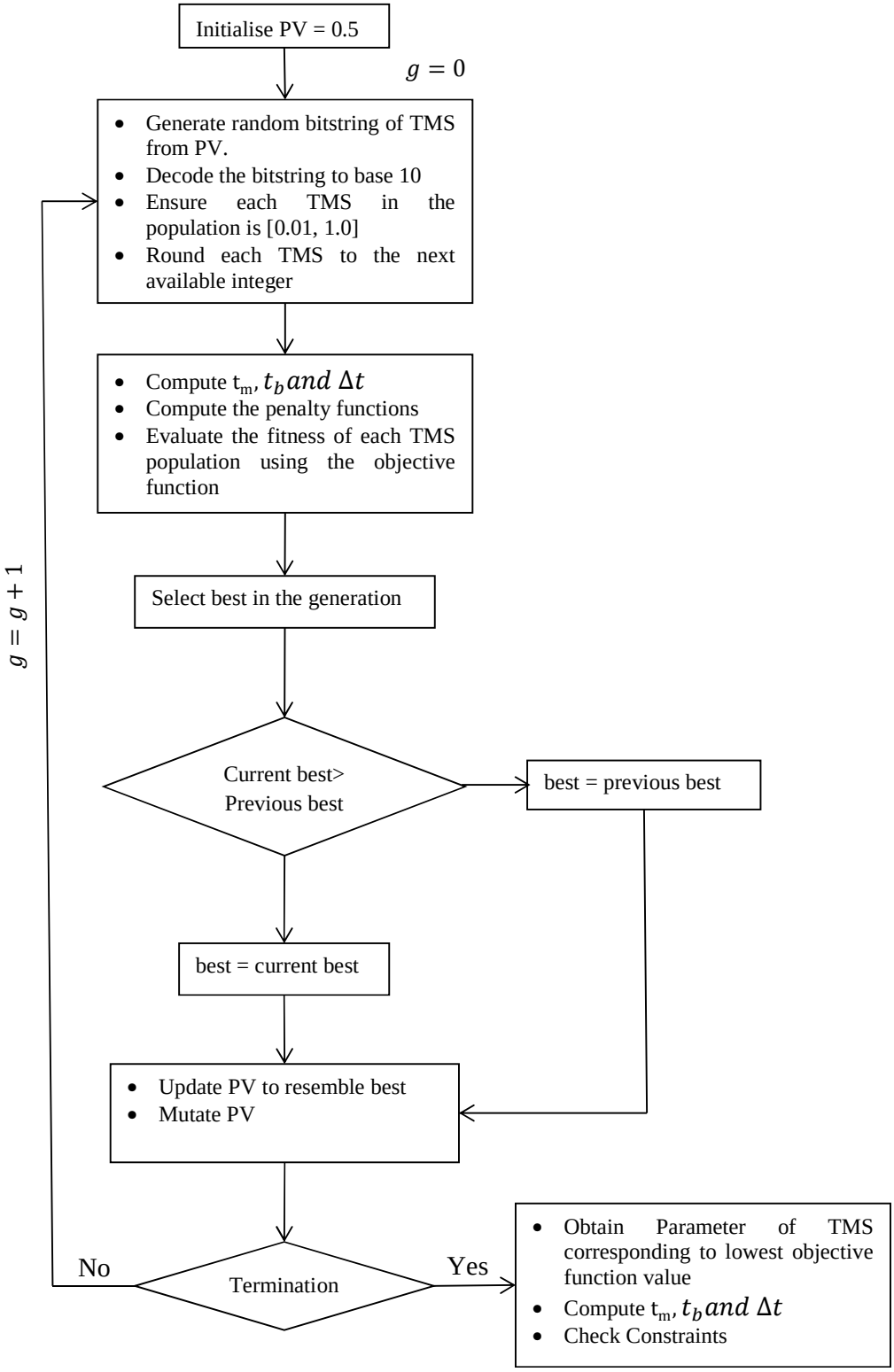

Figure 4.4 Flow chart illustrating Application of PBIL for relay coordination

4.7 Summary

In this chapter, a variety of approaches from previous work on application of evolutionary algorithms are discussed and these are : relay operating characteristics, objective function modelling, and handling of constraints violations as well as representation of variables. This review was necessary to help inform how the selected evolutionary algorithms were applied. The following application decisions were made in this chapter:

- Relay operating characteristic : IEC standard inverse
- Optimization problem modelling: Linear
- Objective function: Considers operating time of the relays.
- Constraint handling: use of penalty functions.

The chapter was concluded by discussing how the selected evolutionary algorithms are applied to solve the relay coordination optimization problem. In the next chapter, the simulation results of the application of evolutionary algorithms for relay coordination will be discussed. The chapter will also discuss the effects of learning rate on PBIL.

CHAPTER 5

5 SIMULATION RESULTS AND DISCUSSIONS

This chapter presents the simulation results and analysis for the IEEE 24 bus system.

5.1 Introduction

As discussed in the previous chapter, the parameters of the overcurrent relays are determined by looking at the following objective function:

- The objective function which minimizes (optimize) the operating time of main relays:

The performance of relay coordination determined through BGA, GA and PBIL evolutionary algorithms is analysed by comparing the resulting fitness function value, operating time of main relays and operating time of back-up relays provided by each algorithm. Another factor that is analysed in the comparison of the evolutionary algorithms is the convergence rate of the algorithms. In addition, the effects of Learning Rate (LR) on the performance of PBIL for relay coordination are analysed.

The results were obtained by running each algorithm ten times. In the comparison of GA, BGA and PBIL the best fitness value of the ten runs was used. In the study of the effects of learning rate on the performance of PBIL, the average fitness value of the ten runs was used. In this research, the termination criterion for the algorithms is the fixed number of generations. The maximum number of generations was chosen by trial and error, through observing that increasing the number of generations beyond 1500 does not result in a significant improvement in terms of performance.

In section 5.2, the power system model used for this research is discussed. In Section 5.3, the results for the three selected evolutionary algorithms are analysed by looking at the fitness value and the convergence rate. The performance of the parameters of overcurrent relays determined by each algorithm is analysed for the first two coordination pairs in Section 5. 4. The performance for the rest of the pairs is given in Table 5.3. The effects of learning rate on the performance of PBIL are discussed in Section 5.5 by looking at different learning rates and different learning rate schemes as discussed in Chapter 3.

5.2 Power System Model used for studies

The IEEE 24 Bus power system is shown in Figure 5.1 [41]. The evolutionary algorithms were applied on the 230kV network of the system which consists of 14 buses, 42 directional overcurrent relays and 100 primary and backup pairs.

Before the evolutionary algorithms are applied for relay coordination, the following has to be done:

- Determine primary and backup pairs for coordination
- Determine primary and backup fault current for each relay pair.
- Determine the pickup current setting based on equipment rating and fault currents.

The results of this process are given in Appendix C. The rated current for all conductors on the 230kV side of the IEEE 24 bus system is 500A. The pickup currents settings for each relay are set to allow for 20% overloading on the conductor rating. Therefore a value of 600A is chosen for all the relays. This value is less than the fault currents measured by the relays for the faults at the remote busbars. Therefore, the pickup current setting is sensitive enough to detect all the faults on the lines. This data is then fed into the evolutionary algorithm to determine optimal TMS parameters for the 42 relays.

5.3 Fitness values, Convergence rate and Overcurrent relay parameters

Figures 5.2-5.5 shows the fitness value of the evolutionary algorithms for the objective function considered. It can be seen that, the GA algorithm converges to the fitness value of 19.83seconds. The BGA and PBIL converged to the fitness value of 18.62 seconds and 48.02 seconds, respectively. These results show that BGA performs slightly better than GA. The PBIL algorithm yields the worst performance due to premature convergence. Since for this problem an optimal solution is not know in advance, the results of the conventional coordination method are used to measure the performance of evolutionary algorithms. Compared to the overall operating time for main relays given by conventional coordination method, BGA and GA algorithms yield satisfactory results. In contrast, the results provided by PBIL algorithm are not satisfactory.

Furthermore, it can be deduced from the results that BGA has the ability to exploit the search space much more efficiently in the beginning of the search. The fitness value for the BGA algorithm drops rapidly from the initial value of 718 seconds to 70 seconds where it begins to settle to a fitness value in 7 iterations. It settles between 70 and 65 seconds for about 28 iterations. The algorithm maintains diversity and converges steadily until it reaches a fitness value that is within 5% of the final fitness value (19.55 seconds) in 1 008 iterations. It reaches the final value of 18.62 seconds in 1 132 iterations.

The GA algorithm starts at the fitness value of 703 seconds and drops to 64 seconds in 49 iterations. It settles to 64 seconds for a few iterations and gradual converges until it reaches the fitness value that is within 5% of the final fitness value (20.80 seconds) in 1119 iterations. It reaches the final fitness value of 19.84 seconds in 1456 iterations.

The PBIL algorithm starts at the fitness value of 814 seconds and reaches a fitness value of 74 seconds in 77 iterations. The algorithm reaches the fitness value that is within 5% of the final fitness value after 397 iterations and slowly converges until it reaches the final fitness value of 48.02 seconds.

The reason for this is that BGA selects only 10% of the best solution that takes part in the evolution deterministically while GA considers 50% of best solutions selected probabilistically in the creation of the new solution and PBIL depends on the learning to create new solutions. However, both the BGA and the GA shows consistent improvement in the fitness value as the search progress until the end of the iterations. In contrast, PBIL converges prematurely after the initial turning point in the beginning of the search.

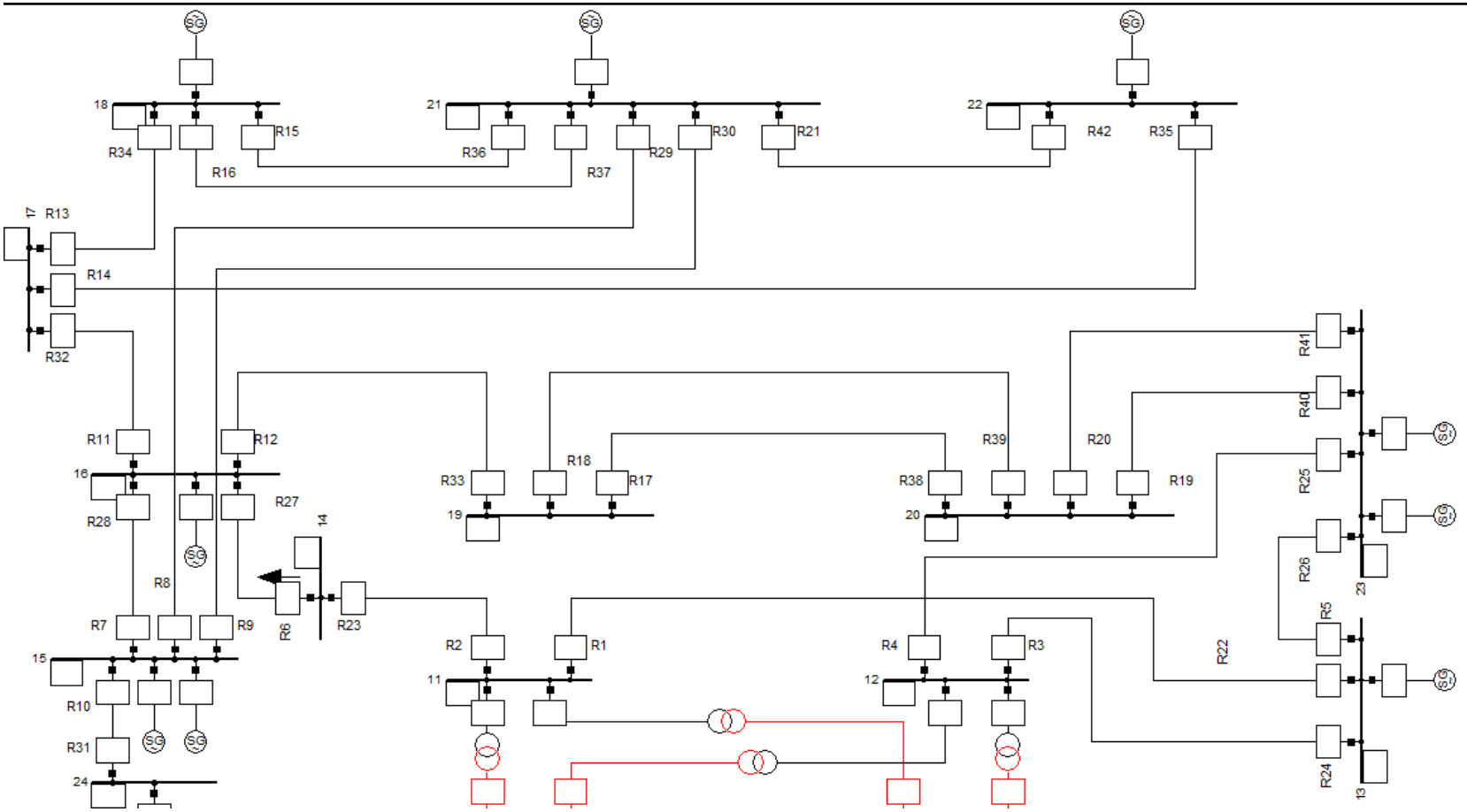

Figure 5.1 The 230kV side of the IEEE 24 Bus system [41]

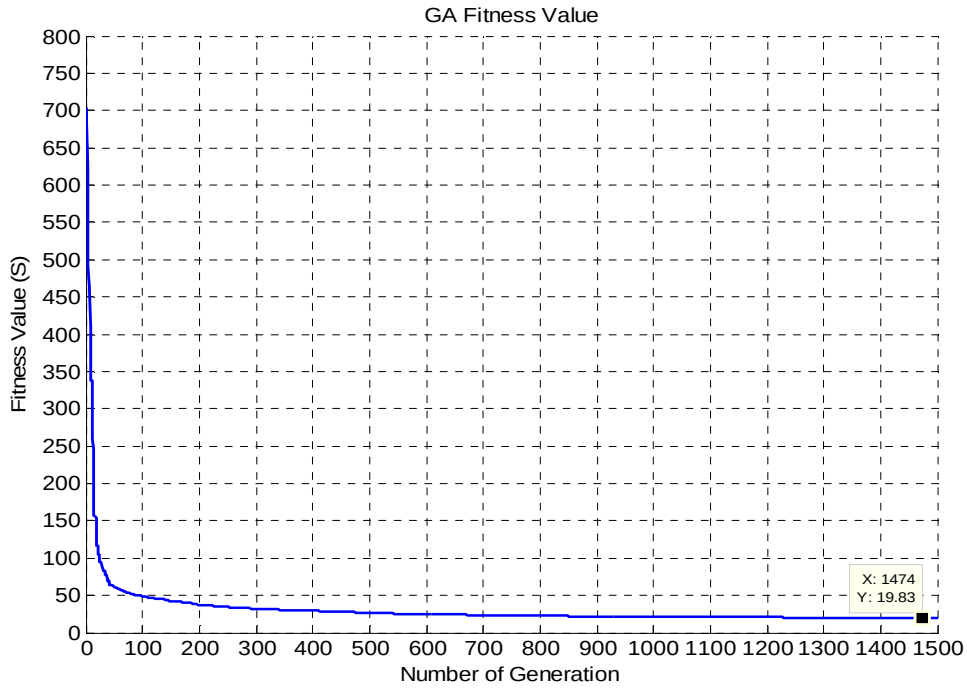

Figure 5.2 Fitness Value for GA

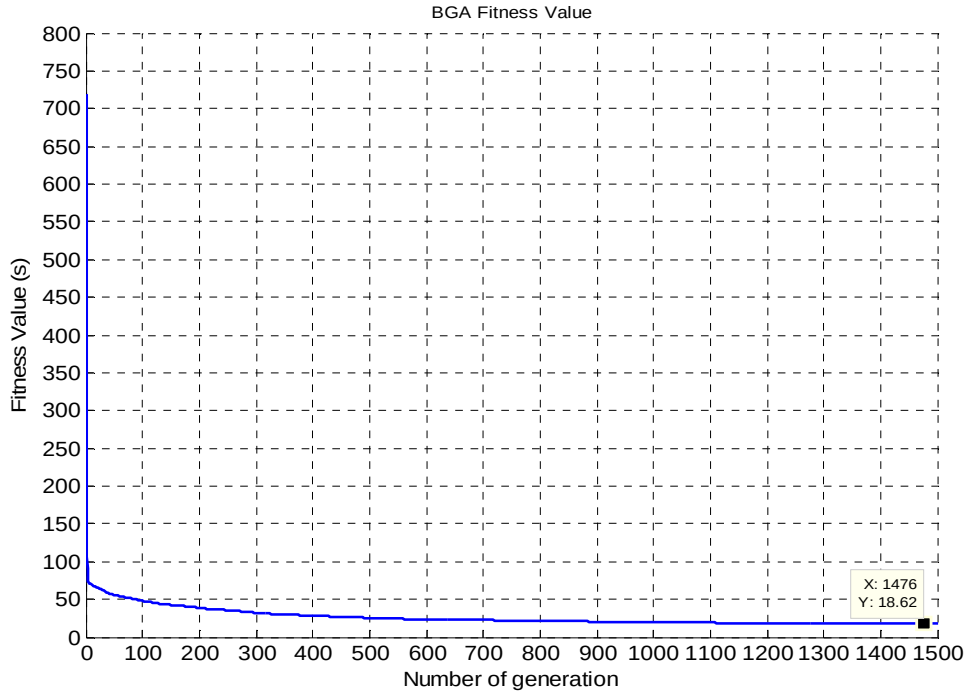

Figure 5.3 Fitness Value for BGA

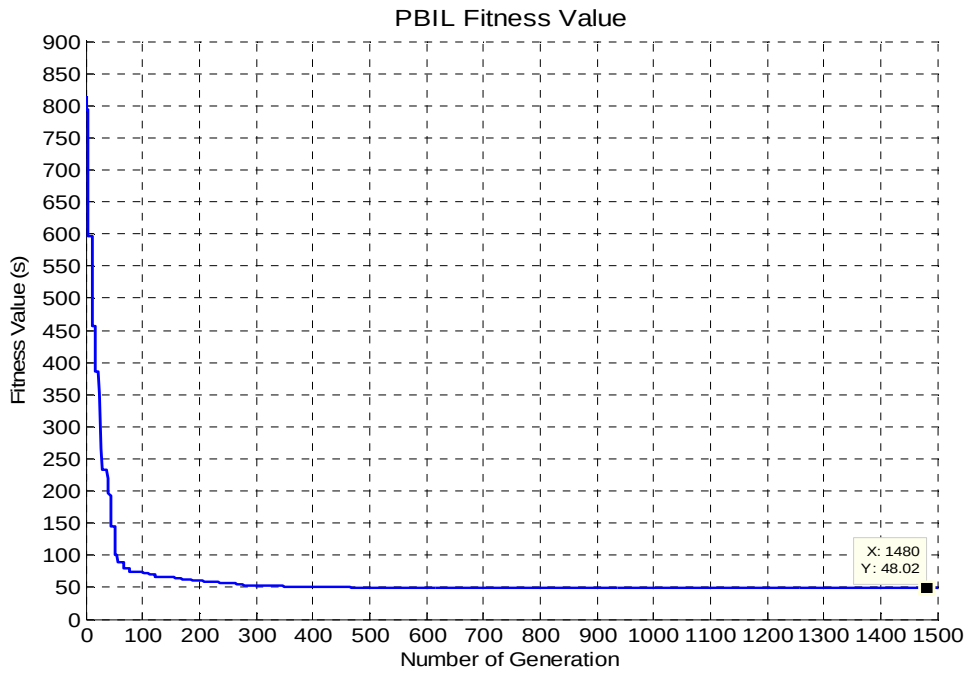

Figure 5.4 Fitness Value for PBIL

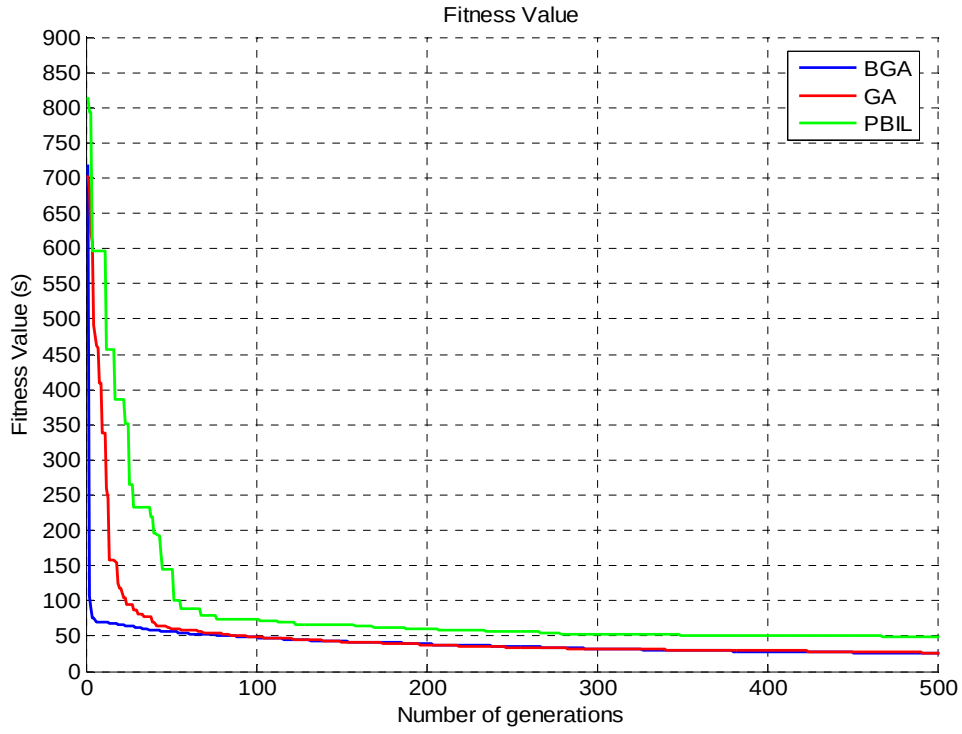

Figure 5.5 Comparison of Fitness Value for GA, BGA and PBIL

The following parameters of overcurrent relays were obtained from each optimization algorithm as indicated in Table 5.1. Also included in the table are the parameters determined from conventional coordination method. It can be seen from Table 5.1 that BGA provides TM parameters of the directional overcurrent relays that are slightly smaller than parameters provided by the GA. Due to premature convergence indicated above; PBIL provides TM parameters which are much higher than the parameters of both GA and BGA. The parameters provided by both GA and BGA are similar to the parameters provided by conventional relay coordination method.

Table 5.1Parameters of overcurrent relay coordination methods

	GA	BGA	PBIL	Conventional
TM 1	0.12	0.12	0.28	0.12
TM 2	0.25	0.23	0.6	0.23
TM 3	0.16	0.15	0.52	0.15
TM 4	0.12	0.11	0.27	0.11
TM 5	0.16	0.14	0.3	0.14
TM 6	0.17	0.15	0.51	0.15
TM 7	0.28	0.26	0.51	0.27
TM 8	0.16	0.15	0.41	0.15
TM 9	0.16	0.15	0.29	0.15
TM 10	0.02	0.01	0.03	0.01
TM 11	0.27	0.26	0.56	0.26
TM 12	0.23	0.22	0.51	0.21
TM 13	0.2	0.19	0.51	0.19
TM 14	0.12	0.11	0.33	0.11
TM 15	0.12	0.16	0.31	0.16
TM 16	0.18	0.16	0.52	0.16
TM 17	0.17	0.14	0.27	0.13

TM 18	0.15	0.14	0.27	0.13
TM 19	0.15	0.11	0.29	0.15
TM 20	0.1	0.09	0.27	0.09
TM 21	0.15	0.13	0.51	0.13
TM 22	0.25	0.23	0.51	0.23
TM 23	0.14	0.14	0.54	0.14
TM 24	0.15	0.15	0.51	0.15
TM 25	0.14	0.14	0.34	0.14
TM 26	0.17	0.16	0.52	0.16
TM 27	0.24	0.23	0.66	0.23
TM 28	0.23	0.21	0.55	0.21
TM 29	0.21	0.19	0.52	0.2
TM 30	0.2	0.19	0.52	0.2
TM 31	0.1	0.09	0.15	0.09
TM 32	0.22	0.2	0.53	0.2
TM 33	0.25	0.22	0.53	0.22
TM 34	0.3	0.28	0.62	0.28
TM 35	0.11	0.1	0.52	0.1
TM 36	0.24	0.23	0.52	0.23
TM 37	0.24	0.23	0.51	0.23
TM 38	0.23	0.21	0.52	0.2
TM 39	0.24	0.21	0.52	0.2
TM 40	0.24	0.22	0.59	0.22
TM 41	0.24	0.22	0.52	0.22
TM 42	0.13	0.13	0.27	0.13

The parameters above yield the following overall operating times for all the relays on the network. The results in Table 5.2 show that BGA provides slightly smaller operating time for main relays and backup relays compared to GA. The results of GA and BGA are similar to results provided by conventional relay coordination method. . PBIL provides the highest overall operating times.

Table 5.2 Overall operating times for overcurrent relay coordination methods

	GA	BGA	PBIL	Conventional
Main operating time ($\sum t_m$)	19.83 s	18.62 s	48.02 s	18.65 s
Backup operating time ($\sum t_b$)	93.46s	86.61 s	216.47	86.98 s

5.4 Relay coordination for system faults

The results in this section are provided to demonstrate the performance of the individual relays for faults in their primary and backup protection zones. The performance is analysed by looking at the grading margin, operating time for primary zone fault and operating time for backup zone fault for each algorithm. Due to a large number of main and backup pairs, only two relay pairs will be analysed. The performance of other relay pairs is given in Table 5.3.

5.4.1 Performance analysis for Relay Pair 1: (Relay 6 and Relay 2)

Figures 5.6 – 5.8 show the coordination curves for the selected relay coordination pair for GA, BGA and PBIL algorithms. The main relay for the selected pair is Relay 6 and the backup relay is Relay 2. For this relay pair, a three phase fault was simulated in front of Relay 6. Both relays measure 3120 A for the fault. For GA, Relay 6 operates in 0.710 seconds and Relay 2 operates in 1.044 seconds. The relays operated properly with the grading margin of 0.334 seconds which is above the coordination time interval of 0.3 seconds. For BGA, Relay 6 operates in 0.626 seconds and Relay 2 operates in 0.961 seconds. The relays operated properly with the grading margin of 0.334 seconds which is above the coordination time interval of 0.3 seconds. For PBIL, Relay 6 operates in 2.130

seconds and Relay 2 operates 2.506 seconds. In terms of coordination, the relays operated properly with the grading margin of 0.376 seconds which is above the coordination time interval of 0.3 seconds. However, the response of relays for the fault is much longer than is the case for GA and BGA. This violates one of the principles of protection which is to isolate a fault from the power system as quickly as possible.

5.4.2 Performance analysis for Relay Pair 2: (Relay 12 and Relay 6)

Figures 5.9 – 5.11 show the coordination curves for the selected relay coordination pair for GA, BGA and PBIL algorithms. The main relay for the selected pair is Relay 12 and the backup relay is Relay 6. For this relay pair, a three phase fault was simulated in front of Relay 12. This fault is in the primary zone of protection for relay 12 and in the backup zone of protection for Relay 6.For this fault Relay 12 measures 12674 A and Relay 6 measures 1854A. For GA, Relay 12 operates in 0.521 seconds and Relay 6 operates in 1.043 seconds. The relays operated properly with the grading margin of 0.522 seconds which is above the coordination time interval of 0.3 seconds. For BGA, Relay 12 operates in 0.499 seconds and Relay 6 operates in 0.920 seconds. The relays operated correctly with the grading margin of 0.422 seconds which is above the coordination time interval of 0.3 seconds. For PBIL, Relay 12 operates in 1.156 seconds and Relay 6 operates 3.130 seconds. In terms of coordination, the relays operated properly with the grading margin of 1.973 seconds which is above the coordination time interval of 0.3 seconds. However, the response of the relays for the fault is much longer than is the case for GA and BGA. This violates one of the principles of protection which is to isolate a fault from the power system as quickly as possible. In a similar manner, Table 5.3 provides the operating times and grading margins for the rest of the coordination pairs. It can be seen that the three evolutionary algorithms provides coordination for all relay pairs. However, in general, for PBIL the response of the relays is much longer than for both the GA and BGA.

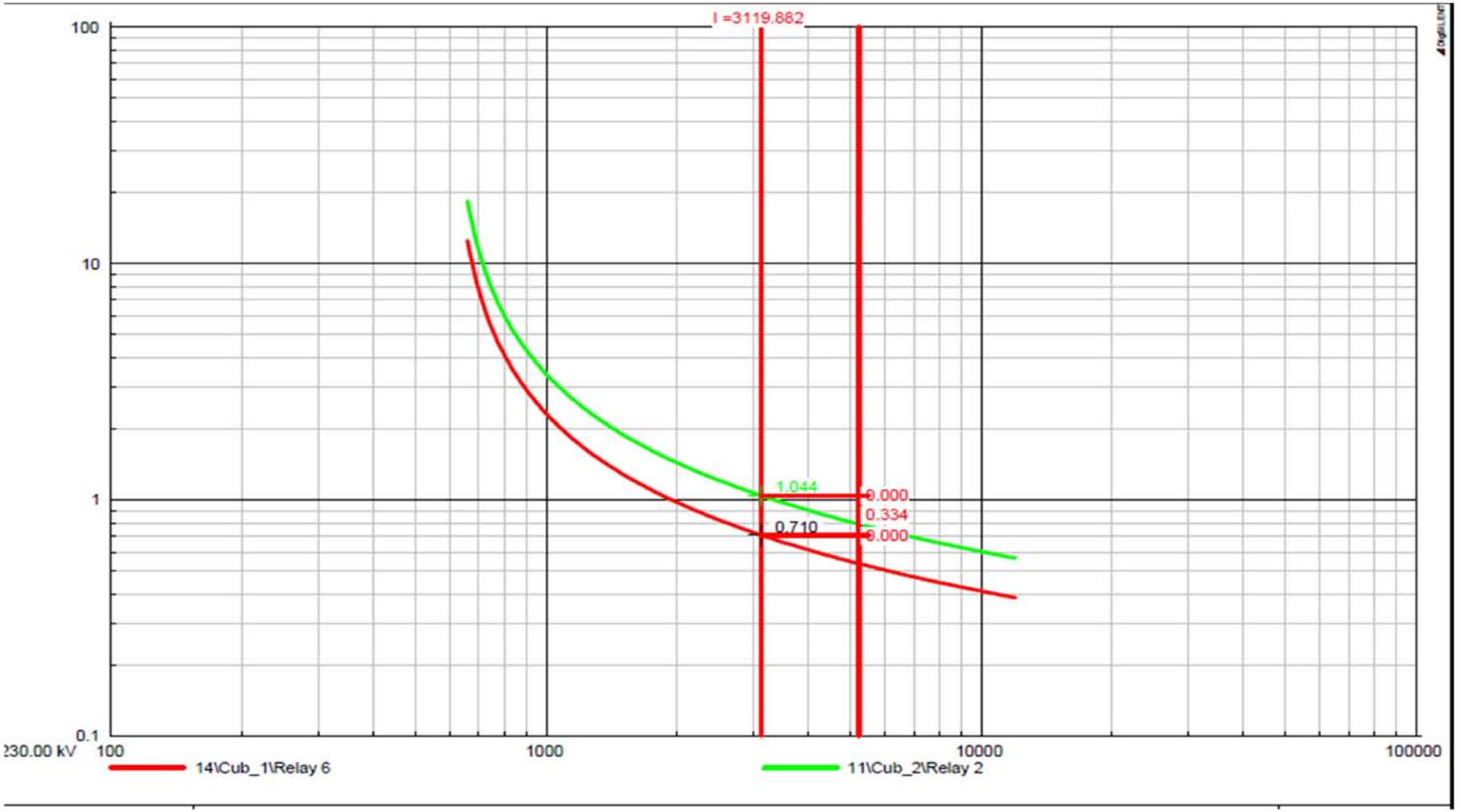

Figure 5.6 Performance of GA for Relay Pair 1

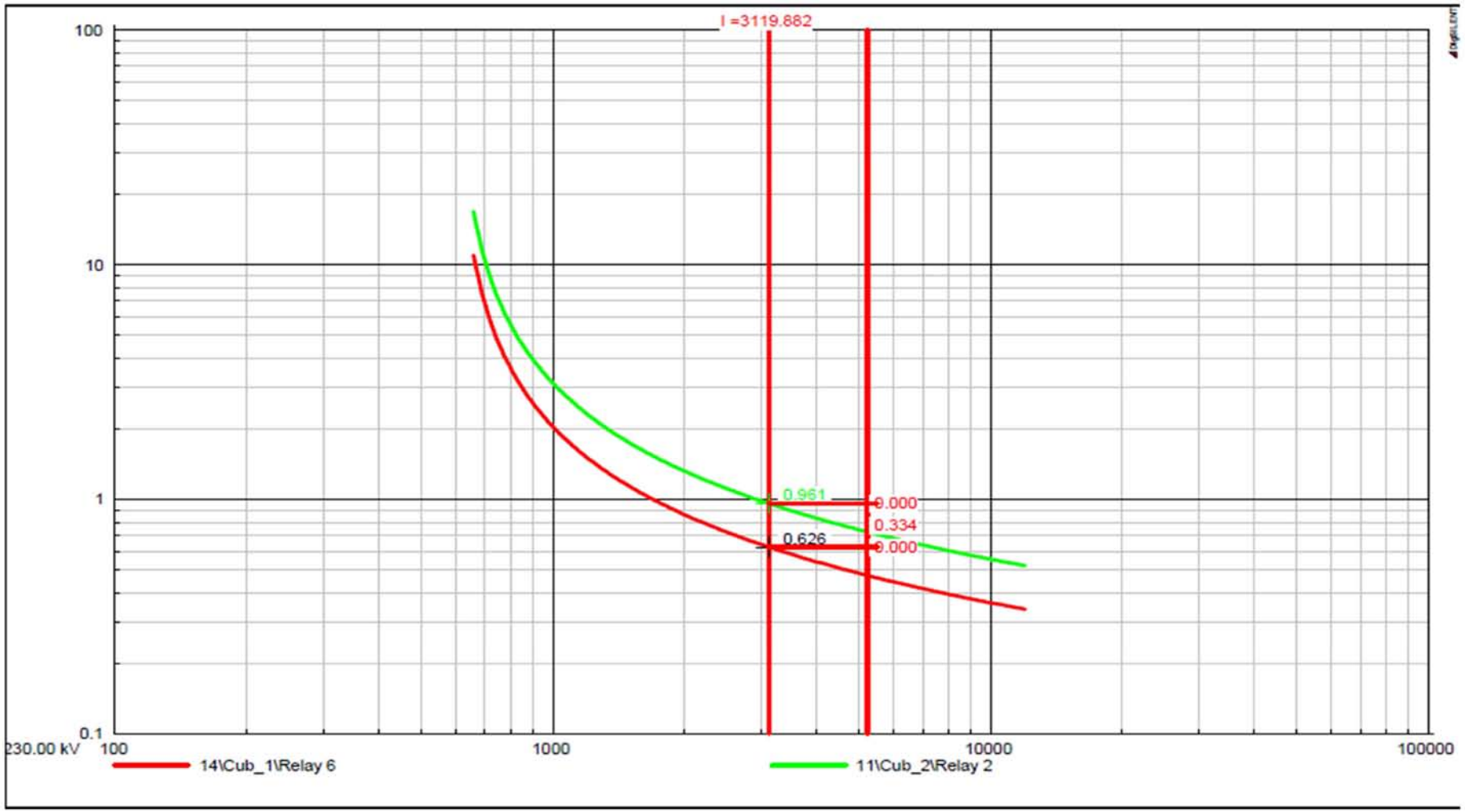

Figure 5.7 Performance of BGA for Relay Pair 1

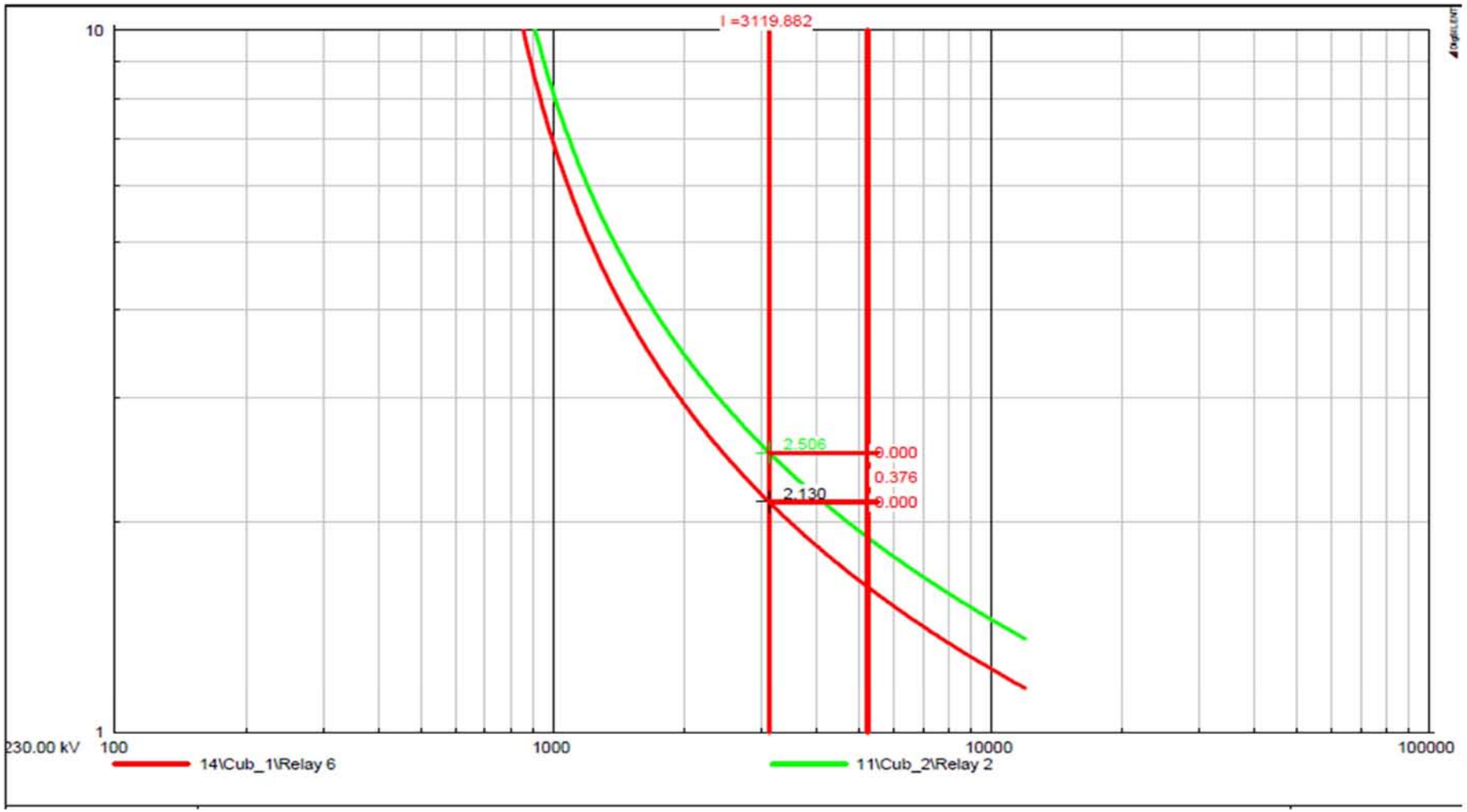

Figure 5.8 Performance of PBIL for Relay Pair 1

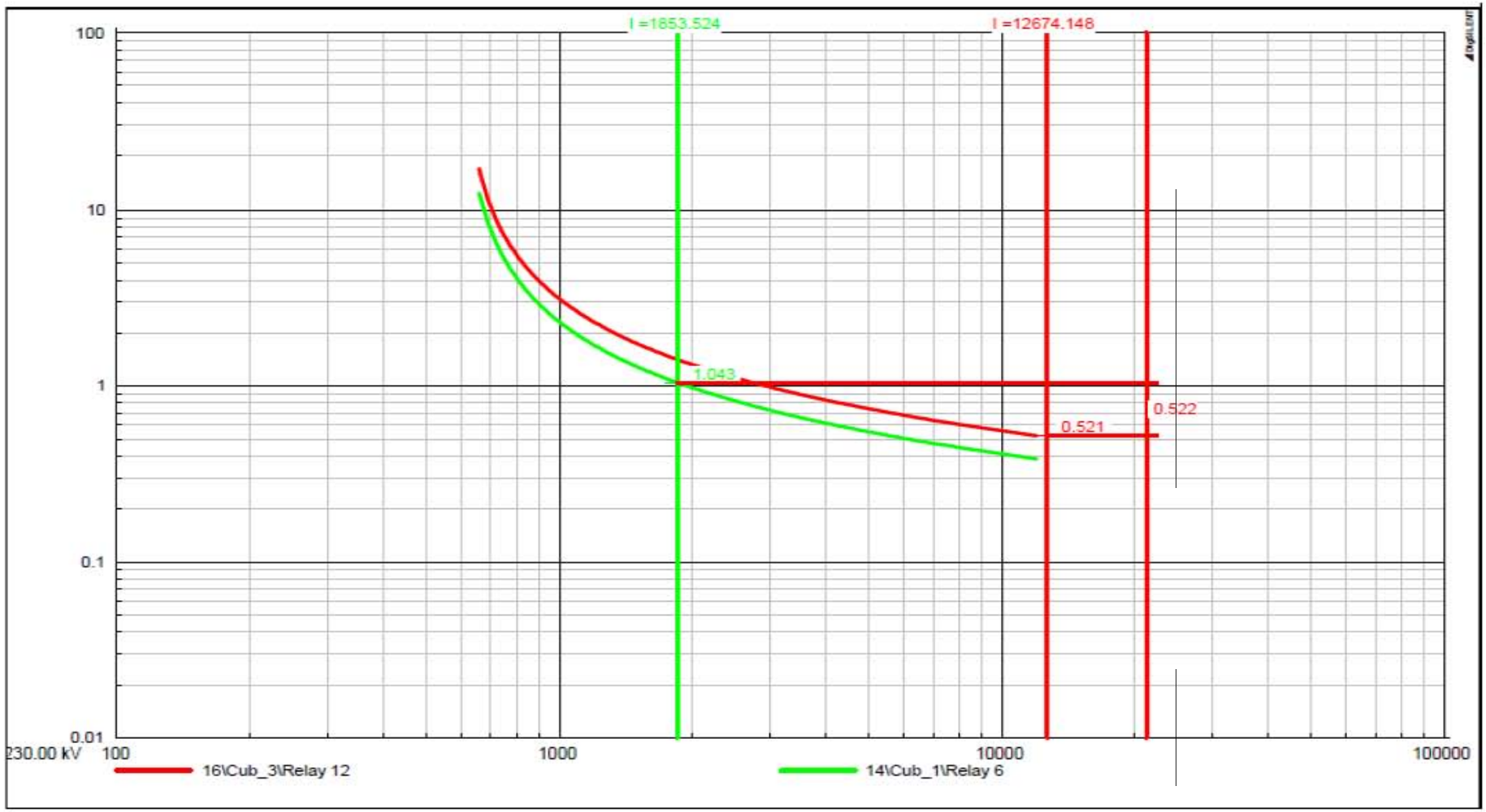

Figure 5.9 Performance of GA for Relay Pair 2

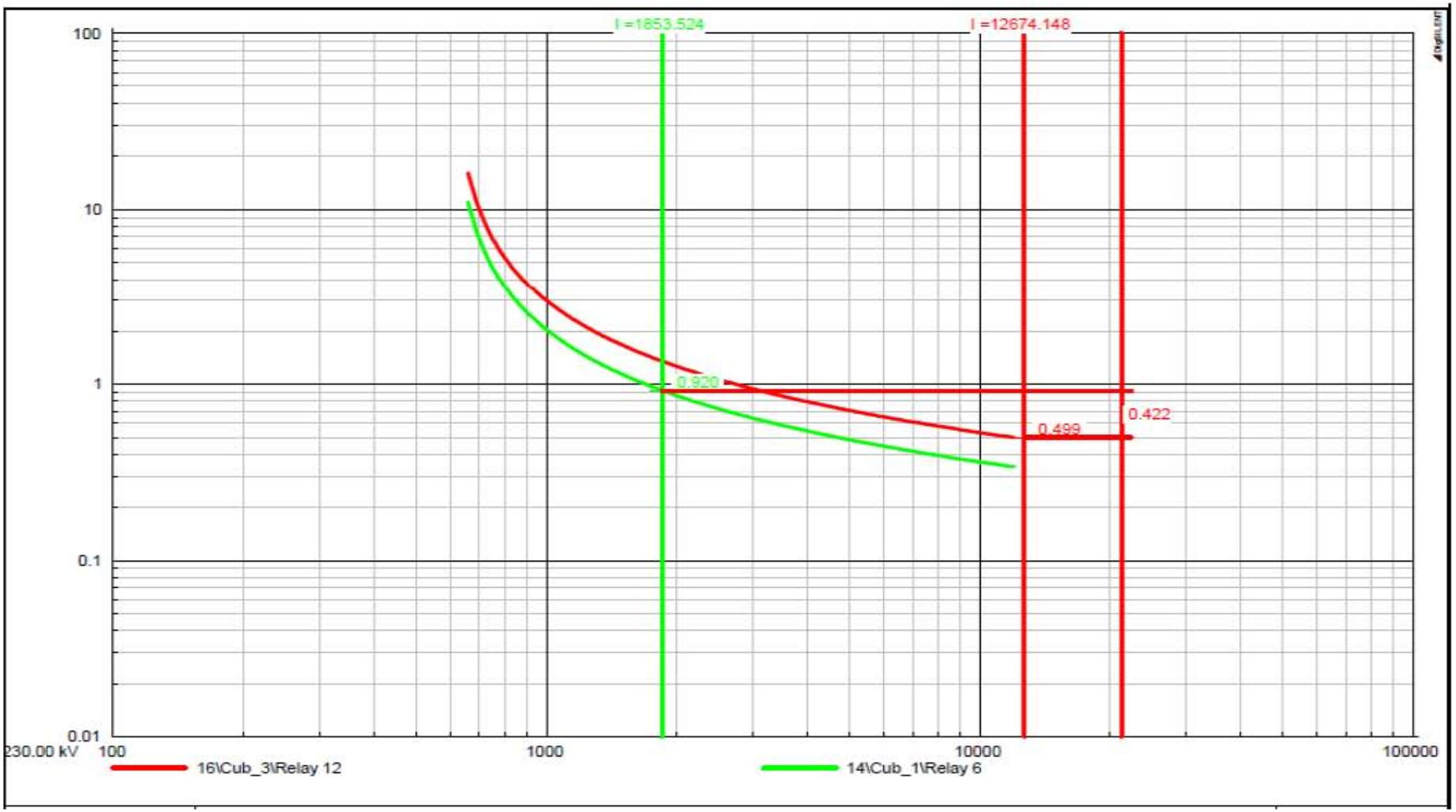

Figure 5.10 Performance of BGA for Relay Pair 2

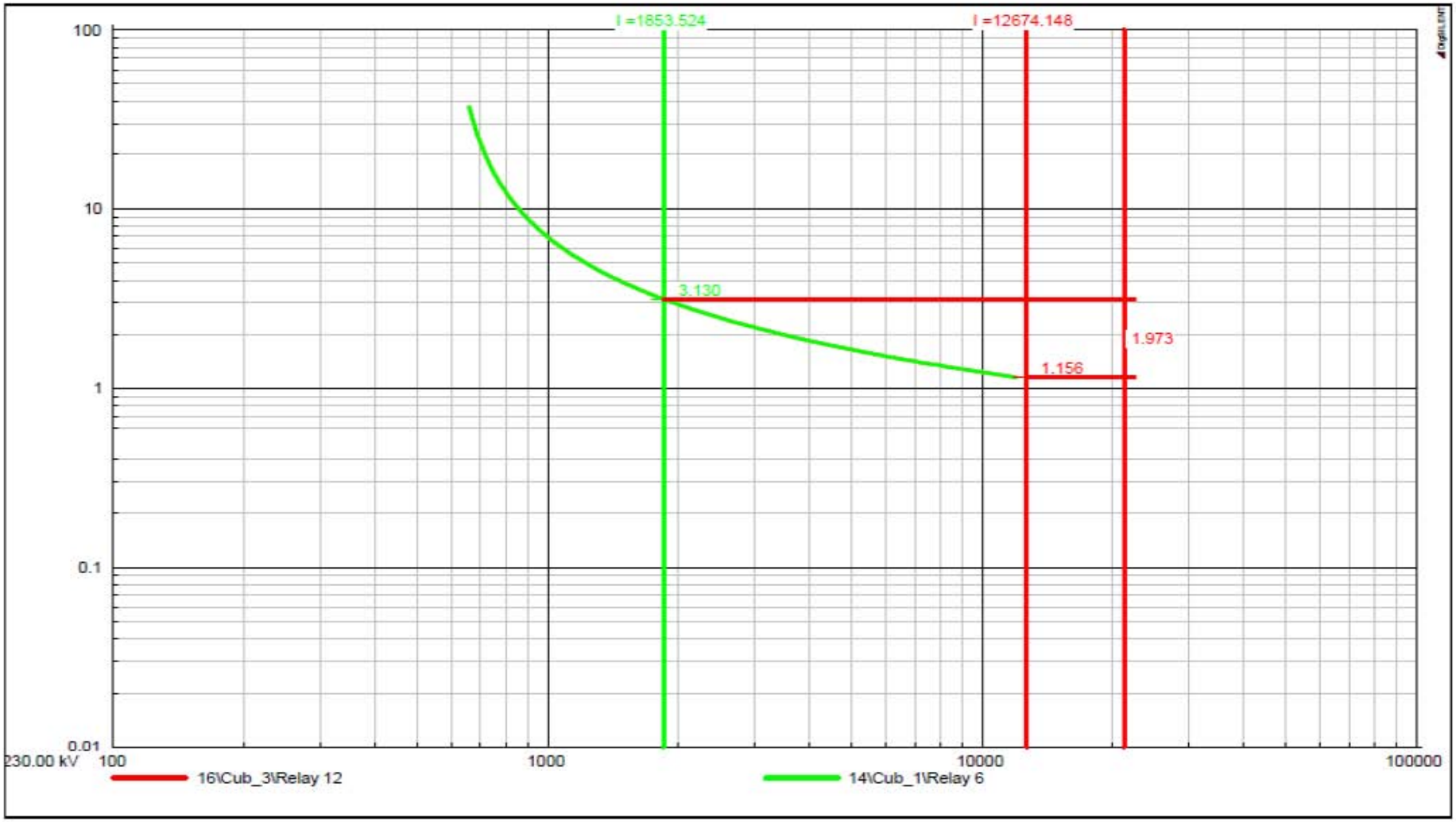

Figure 5.11 Performance of PBIL for Relay Pair 2

Table 5.3 The operating times of Evolutionary Algorithms for the objective function: main operating time

	BGA			GA			PBIL		
	t_m(s)	t_b(s)	Δt(s)	t_m(s)	t_b(s)	Δt(s)	t_m(s)	t_b(s)	Δt(s)
1 – 23	0.350	0.699	0.045	0.383	0.699	0.016	0.825	2.696	1.844
2 – 22	0.645	0.962	0.017	0.702	1.047	0.045	1.684	2.135	0.151
3– 25	0.452	0.807	0.056	0.482	0.808	0.026	1.566	1.962	0.096
4 – 24	0.311	0.642	0.03	0.339	0.642	0.003	0.763	2.183	1.120
5– 1	0.338	0.669	0.03	0.386	0.725	0.039	0.724	1.562	0.538
5 – 3		0.890	0.252		0.949	0.264		3.086	2.062
6 – 2	0.616	0.944	0.028	0.698	1.027	0.029	2.095	2.465	0.070
7 – 29	0.688	1.011	0.023	0.741	1.118	0.077	1.349	2.768	1.119
7 – 30		1.011	0.023		1.064	0.024		2.768	1.1119
7 – 31		1.039	0.052		1.155	0.114		1.732	0.083
8 – 28	0.354	0.668	0.013	0.378	0.731	0.053	0.969	1.749	0.480
8 – 30		1.056	0.41		1.112	0.434		2.89	1.621
8 – 31		1.088	0.434		1.209	0.531		1.813	0.544
9 – 28	0.354	0.669	0.013	0.378	0.731	0.053	0.685	1.749	0.764
9 – 29		1.056	0.401		1.167	0.489		2.891	1.906
9 – 31		1.088	0.434		1.209	0.531		1.813	0.828
10 – 28	0.023	0.668	0.346	0.046	0.732	0.387	0.069	1.751	1.382
10 – 29		1.036	0.714		1.146	0.800		2.837	2.468

10 – 30		1.036	0.714		1.091	0.745		2.837	2.468
11 – 6	0.605	0.909	0.0.04	0.629	1.030	0.101	1.304	3.090	1.486
11 – 7		0.905	0.0004		0.975	0.045		1.775	0.171
11 – 33		0.909	0.004		1.033	0.105		2.191	0.587
12 – 6	0.511	0.909	0.098	0.534	1.030	0.196	1.184	3.092	1.608
12 – 7		0.900	0.089		0.969	0.135		1.765	0.281
12 – 32		0.8200	0.009		0.902	0.068		2.173	0.689
13 – 11	0.564	0.866	0.002	0.593	0.899	0.06	1.512	1.865	0.053
13 – 35		0.889	0.025		0.978	0.085		4.625	2.813
14 – 11	0.270	0.897	0.326	0.295	0.931	0.336	0.811	1.932	0.821
14 – 34		0.889	0.318		0.952	0.357		1.968	0.857
15 – 13	0.386	0.711	0.026	0.434	0.7488	0.015	0.747	1.909	0.862
15 – 37		1.042	0.356		1.087	0.354		2.311	1.264
16 – 13	0.386	0.711	0.025	0.410	0.749	0.039	1.253	1.909	0.356
16 – 36		1.042	0.357		1.088	0.378		2.356	0.983
17 – 12	0.371	0.678	0.005	0.398	0.707	0.009	0.716	1.569	0.553
17 – 39		1.081	0.410		1.236	0.538		2.679	1.663
18 – 12	0.371	0.677	0.005	0.398	0.707	0.009	0.7163	1.569	0.553
18 – 38		1.081	0.4103		1.185	0.487		2.679	1.663
19 – 17	0.296	0.830	0.234	0.296	0.889	0.293	0.781	1.601	0.521

19 – 18		0.830	0.234		0.889	0.293		1.601	0.521
19 – 1		0.829	0.233		0.904	0.308		1.959	0.878
20 – 17	0.242	0.830	0.287	0.296	0.889	0.320	0.727	1.601	0.574
20 – 18		0.830	0.287		0.889	0.320		1.601	0.574
20 – 40		0.828	0.287		0.904	0.335		2.223	1.196
21 – 8	0.297	0.893	0.296	0.343	0.953	0.310	1.165	2.442	0.977
21 – 9		0.893	0.296		0.953	0.310		1.727	0.262
21 – 15		0.845	0.248		0.951	0.307		1.638	0.173
21 – 16		0.845	0.248		0.898	0.255		2.747	1.282
22 – 3	0.551	0.873	0.023	0.599	0.932	0.033	1.221	3.028	1.507
22 – 26		0.901	0.051		0.958	0.059		2.931	1.410
23 – 27	0.487	0.799	0.012	0.487	0.834	0.047	1.877	2.294	0.117
24 – 1	0.358	0.659	0.002	0.358	0.715	0.057	1.216	1.539	0.023
24 – 26		0.900	0.243		0.957	0.299		2.927	1.411
25 – 5	0.319	0.880	0.261	0.319	1.006	0.386	0.775	1.885	0.810
25 – 19		0.839	0.220		0.839	0.220		2.213	1.138
25 – 20		0.686	0.068		0.763	0.144		2.061	0.986
26 – 4	0.368	0.883	0.216	0.391	0.964	0.273	1.195	2.170	0.675
26 – 19		0.827	0.160		0.828	0.137		2.182	0.687
26 – 20		0.677	0.009		0.753	0.062		2.032	0.537

27 – 7	0.517	0.9084	0.091	0.540	0.978	0.138	1.484	1.782	0.000
27 – 32		0.828	1.011		0.911	0.072		2.195	0.411
27 – 33		0.918	0.102		1.044	0.205		2.214	0.430
28 – 6	0.507	0.903	0.096	0.555	1.023	0.168	1.326	3.070	1.444
28 – 32		0.821	0.014		0.903	0.048		2.175	0.549
28 – 33		0.904	0.098		1.028	0.173		2.180	0.554
29 – 9	0.437	0.907	0.169	0.483	0.967	0.184	1.195	1.753	0.258
29 – 15		0.833	0.096		0.937	0.154		1.614	0.119
29– 16		0.833	0.096		0.885	0.102		2.708	1.213
29 – 42		0.9088	0.172		0.9088	0.126		1.888	0.393
30 – 8	0.437	0.907	0.169	0.460	0.967	0.207	1.195	2.478	0.983
30 – 15		0.833	0.096		0.937	0.177		1.614	0.119
30 – 16		0.833	0.096		0.885	0.125		2.708	1.213
30 – 42		0.908	0.172		0.908	0.149		1.888	0.393
31- none	0.612			0.617			1.195		
32 – 34	0.561	0.8614	0.000	0.617	0.923	0.006	1.485	1.908	0.123
32 – 35		0.885	0.025		0.975	0.057		4.607	2.822
33 – 38	0.726	1.048	0.021	0.826	1.148	0.022	1.750	2.595	0.545
33 – 39		1.048	0.021		1.198	0.072		2.595	0.545
34 – 36	0.695	1.017	0.022	0.745	1.062	0.017	1.539	2.300	0.461

34 – 37		10.17	0.022		1.062	0.017		2.257	0.418
35 – 21	0.302	0.611	0.009	0.332	0.705	0.073	1.572	2.398	0.526
36 – 8	0.530	0.866	0.036	0.553	0.923	0.070	1.199	2.367	0.868
36 – 9		0.866	0.036		0.923	0.070		1.674	0.175
36 – 16		0.843	0.013		0.896	0.042		2.740	1.241
36 – 42		0.892	0.062		0.892	0.038		1.853	0.0.354
37 – 8	0.530	0.866	0.036	0.553	0.923	0.070	1.176	2.367	0.891
37 – 9		0.866	0.036		0.923	0.0070		1.674	0.198
37 – 15		0.843	0.012		0.949	0.095		1.634	0.270
37 – 42		0.892	0.062		0.892	0.039		1.853	0.377
38 – 18	0.522	0.873	0.051	0.572	0.936	0.064	1.292	1.684	0.092
38 – 40		0.826	0.004		0.902	0.030		2.216	0.624
38 – 41		0.826	0.004		0.902	0.030		1.953	0.361
39 – 17	0.522	0.873	0.052	0.596	0.936	0.039	1.292	1.684	0.092
39 – 40		0.826	0.004		0.902	0.005		2.216	0.624
39 – 41		0.826	0.004		0.902	0.005		1.953	0.361
40 – 4	0.274	0.813	0.239	0.298	0.887	0.289	0.734	1.996	0.962
40 – 5		0.817	0.243		0.934	0.335		1.751	0.717
40 – 20		0.692	0.119		0.769	0.171		2.077	1.043
41 – 4	0.492	0.813	0.021	0.537	0.887	0.005	1.164	1.996	0.532

41 – 5		0.817	0.024		0.934	0.097		1.751	0.287
41 -19		0.846	0.054		0.846	0.009		2.232	0.768
42 - 14	0.422	0.732	0.009	0.422	0.799	0.076	0.878	2.196	1.018

5.5 Effects of Learning Rate (LR) on the Performance of PBIL

The effects of the learning rate on the performance of relay coordination for overcurrent relay parameters determined through PBIL is analysed by comparing the resulting average fitness function value for 10 trials and convergence rate. Another factor that was analysed is the success rate of each learning rate scheme. The success rate measures the number of times a learning rate scheme provides feasible solutions over the total number of times the algorithm was run. For illustration purposes, 600 generations are used to study the effects of learning rate on the performance of the PBIL algorithm. The results of the algorithm do not change significantly after 600 generations.

Figure 5.12 and Figure 5.13 show the convergence rate of the average of the best fitness values over 10 trials for the different learning rates tested on each learning scheme as explained above. Table 5.4 and Table 5.5 show the final fitness value and the success rate for each learning rate. Therefore, Figures 5.12 and 5.13 must be read in conjunction with Tables 5.4 and 5.5.

5.5.1 Effects of LR for Fixed Learning Rate scheme

Figure 5.12 and Table 5.4 below show the average fitness values over 10 trials for the selected fixed learning rates (LR =0.01, LR = 0.05, LR = 0.1, LR = 0.2, LR = 0.4, LR = 0.6, LR = 0.8 and LR = 1.0). It can be seen that for LR = 0.01 the algorithm attains an average value of 210 seconds after 600iterations. It can also be seen that the algorithm fails to provide a feasible solution out of 10 trials. At LR = 0.05 the algorithm converges to a fitness value of 66.70 seconds after approximately 250 iterations and provide a feasible solution 6 times out of 10 trials. At LR = 0.1 the algorithm converges to an average fitness value of 51.68s after 200 iterations and manages to provide a feasible solution for all the 10 trials. At LR = 0.2 the algorithm converges to a lower fitness value of 45.62 seconds after about 150 iterations. The success rate of the algorithm at LR = 0.2 is lower with the algorithm providing a feasible solution 8 times of 10 trials. At LR = 0.4 the algorithm converges to a fitness value of 43.81 seconds after 150 iterations. At LR = 0.6the algorithm converges to a fitness value of 45.67 seconds, at LR = 0.8the algorithm converges to the fitness value of 49.03 seconds and at LR =1.0 the algorithm converges to the fitness value of 49.03 seconds. These results show that for the fixed learning rate case at lower values of learning rate the algorithm tends to explore the search space much more and requires more generations to converge. In addition, increasing the value of LR improves the ability of the

algorithm to converge. However, this also increases the possibility of the algorithm to convergence prematurely. Furthermore, the results show that for a fixed learning rate scheme a good trade-off between the algorithm's ability to converge (exploitation) and the algorithm's ability to search the space widely (exploration) is provided by LR = 0.4 followed by LR = 0.2.

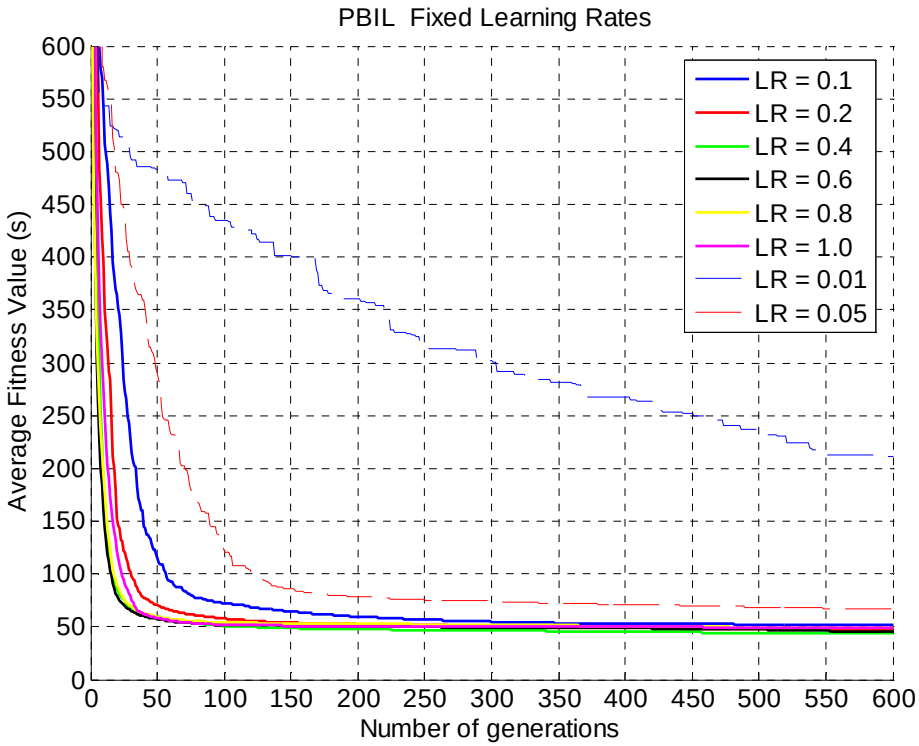

Figure 5.12 Effects of Fixed Learning Rate on PBIL

Table 5.4 shows the final average fitness value and the success rate for the PBIL algorithm for different values of the fixed learning rate scheme.

Table 5.4 Average fitness values and success rate for fixed learning rate

LR:	0.01	0.05	0.1	0.2	0.4	0.6	0.8	1.0
Average Fitness (s)	210	66.79	51.68	45.62	43.81	45.67	49.03	49.24
Success Rate (%)	0	60	100	80	90	80	60	60

5.5.2 Effects of LR for Adaptive Learning Rate scheme

Figure 5.13 and Table 5.5 show the average fitness values over 10 trials for the selected adaptive learning rates (LR =0.01, LR = 0.05, LR = 0.1, LR = 0.2, LR = 0.4, LR = 0.6, LR = 0.8 and LR = 1.0). It can be seen that for LR = 0.01 the algorithm attains an average value of 301.30 seconds after 600iterations. It can also be seen that at this value of LR the algorithm fails to converge and does not provide a feasible solution out of 10 trials. At LR = 0.05 the algorithm attains an average fitness value of 77.82 seconds and provides feasible solutions 3 out of 10 trials. At LR = 0.1 the algorithm converges to an average fitness value of 59.52secondsand starts to settle to a final value at after about 350 iterations and manages to provide a feasible solution for all the 10 trials. At LR = 0.2 the algorithm reaches an average fitness value of 50.79seconds after about 300 iterations. At LR = 0.4 the algorithm converges to an average fitness value of 48.25 seconds. At LR = 0.6 the algorithm converges to an average fitness value of 47.651seconds. At LR = 0.8 the algorithm converges to an average fitness value of 47.93 seconds At LR = 1.0 the algorithm converges to the average fitness value of 48.34 seconds. The learning rate values of LR = 0.1, LR = 0.2, LR = 0.4, LR = 0.6, LR = 0.8, LR =1.0 provides the success rate of 100% for all the 10 trials. These results show that for the adaptive learning rate case at lower values of learning rate the algorithm tends to explore the search space much more and requires more generations for convergence. In addition, due to the adaption introduced the algorithm explores the search space much more since the value of LR stars at a very small value and increases with the number of generations. The results show that, contrary to the fixed learning rate case, although increasing the value of LR improves the ability of the algorithm to converge it does not necessarily result in the possibility of the algorithm converging prematurely at high learning rates. This is due to the adaption that is introduced in the beginning of the search. Since the search is started at smaller values and increases to higher values with the number of generations the algorithm is able to explore the search space broadly in the beginning and effectively exploit the best results found so far later in the search. The results also show that the learning rate does not improve the performance of the algorithm to match or be better than the performance of GA, BGA and conventional relay coordination methods.

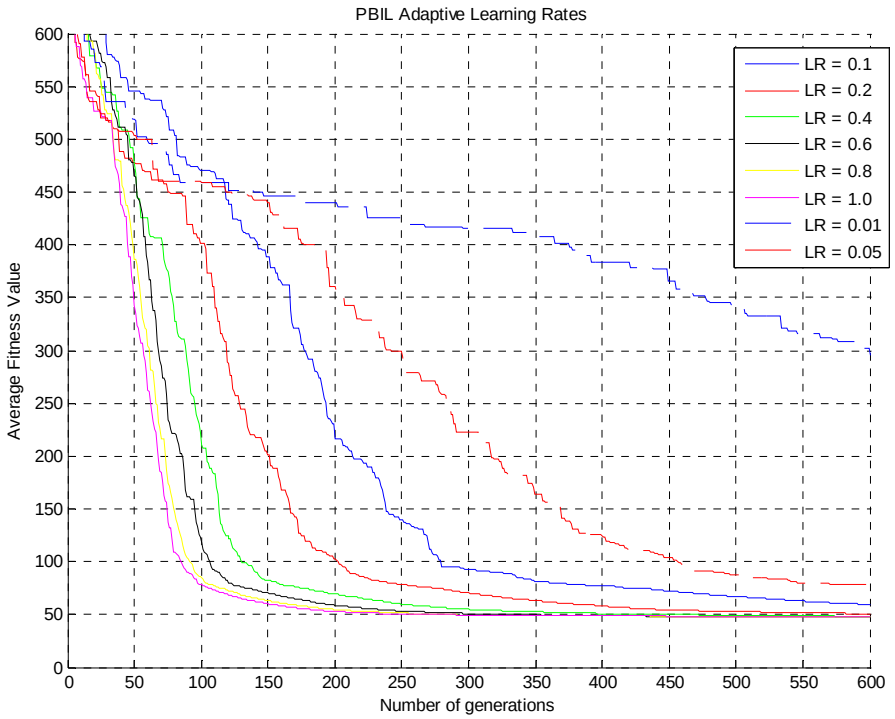

Figure 5.13Effects of Adaptive Learning Rate on PBIL

The table below shows the final average fitness value and the success rate for the PBIL algorithm for different values of the adaptive learning rate scheme.

Table 5.5Average fitness value and success rate for adaptive learning rate.

LR:	0.01	0.05	0.1	0.2	0.4	0.6	0.8	1.0
Average Fitness (s)	301.30	77.82	59.52	50.79	48.25	47.65	47.93	48.34
Success Rate (%)	0	30	100	100	100	100	100	100

5.6 Summary

In this chapter simulation results were analysed and selected algorithms compared. In the next chapter conclusions will be drawn based on the results.

CHAPTER 6

6 CONCLUSIONS

Based on the simulation results obtained in this research, the following conclusions are drawn:

- Evolutionary Algorithms were able to provide settings for the directional overcurrent relays which maintain selectivity or coordination for the 230kV network of the IEEE 24 bus system.

- Since the optimal solution for this problem is not known in advance, the performance of the evolutionary algorithms was compared to the conventional relay coordination method. BGA provides settings that are the same as the conventional relay coordination method. GA provides settings that a slightly bigger than the conventional method while PBIL providing settings which results in the longest operating times. This is due to premature convergence of the PBIL algorithm in the middle of the search.

- It is also observed from the simulation results that BGA converges faster than both the GA and PBIL algorithms in the beginning of the search while PBIL fails to maintain diversity in the middle of the search.

- New trial solutions are resampled from the updated Probability Vector every time the PBIL algorithm goes through the iterations. Therefore, PBIL needs more computational time to carry out the task of optimizing relay coordination on the studied network.

- The simulation results show that the learning rate has an impact on the performance of the PBIL. At lower values of learning rate LR = 0.01 the algorithm explore the search space much more and does not converge. At LR = 0.05, the algorithm converges after many generations and struggles to provide a feasible solution. At higher learning rates the algorithm has the ability to make use of the search results and converges faster. At LR = (0.6-1.0) the algorithm consistently converged to higher values.

- Simulation results also show that introducing adaption on the learning rate of PBIL improves the performance of the algorithm in terms of exploring the search space extensively at the beginning of the search. However, the adaption scheme fails to introduce diversity in the middle of the search and therefore does not improve the performance of PBIL to be better than the performance of both BGA and GA.

CHAPTER 7

7 RECOMMENDATIONS

Based on the conclusions made in this research, the following recommendations are made:

- Both the GA and BGA converge fast in the beginning of the search and do not yield better results than conventional relay coordination method, the possibility of premature convergence in the beginning for both the algorithms can be investigated. For BGA the investigation can focus on the effects of the percentage of top solution truncated to take part in recombination and also looking at the number of solutions that are divided into two halves used to adapt the mutation rate.

- In this research, Evolutionary Algorithms are compared to conventional relay coordination method. Further studies can be conducted to compare Evolutionary Algorithms with other algorithms which have been applied successfully to relay coordination in the past, such as Particle Swarm Optimization (PSO).

- In this research, only the effects of the learning rate on the performance of PBIL were studied. The implementation of different learning rates did not result in the performance that is better than conventional relay coordination. Therefore, an investigation into the effects of other parameters such as population size, mutation scheme and the manner in which the PV is updated can be done to improve the performance of PBIL.

- A further investigation into the effects of crossover method and probability rate as well as mutation method and probability rate on the performance of both the GA and BGA needs to be conducted.

- In this research, the problem of relay coordination was simplified to be linear. Further investigation on the application of GA, BGA and PBIL in which the optimization problem for relay coordination is formulated as nonlinear where both the pickup current and time multiplier settings are determined needs to be done.

- In determining the parameters of the directional overcurrent relay for selected evolutionary algorithms, only the fixed topology of the power system network was

considered. Therefore, another area that can be investigated is to consider different network topologies such that the parameters determined are robust for all power system network contingencies.

- The 230kV network of the IEEE 24 Bus system used in this research is mainly used for reliability studies. To further make a case for application of GA, BGA and PBIL and EAs in general for optimization of relay coordination problem, an implementation on the real utility network could be made and a comparison with the existing settings made.

- Due to the successful application of PBIL and BGA in other power systems related problems, they were applied on the problem of relay coordination in this research. Further investigations in the validity of these algorithms can be done by applying them on benchmark problems and comparing the result with reported results by GA on similar benchmark problems.

REFERENCES

[1]. *IEEE Recommended Practice for Calculating Short-Circuit Currents in Industrial and Commercial Power Systems*, IEEE Standard 551, 2006.

[2]. *IEEE Guide for Protective Relay Applications to Distribution Lines*, IEEE Standard C37.230, 2008.

[3]. Electricity Training Association and Institution of Electrical Engineers, *Power System Protection*. London, United Kingdom: Institution of Electrical Engineers, 1995.

[4]. *IEEE Recommended Practice for Protection and Coordination of Industrial and Commercial Power Systems* ,IEEE Standard 242, 2001.

[5]. *The Authoritative Dictionary of IEEE Standards Terms*, IEEE Standard 100, 2000.

[6]. P. M. Anderson, *Power System Protection*. New York: McGraw-Hill, 1999.

[7]. H.Y. Tsien, "An Automatic Digital Computer Program for Setting Transmission Line Directional Overcurrent Relays," *IEEE Trans. Power App. Syst.*, vol.83, pp. 1048-1053, Oct. 1964.

[8]. R.E. Albrecht et al., "Digital Computer Protective Device Co-ordination Program I-General Program Description," *IEEE Trans. Power App. Syst.*, vol. 83, pp. 402-410, Apr. 1964.

[9]. A.J. Urdaneta et al., "Optimal coordination of directional overcurrent relays in interconnected power systems", *IEEE Trans. Power Del.*, vol. 3, pp. 903-911, Jul. 1988.

[10]. D. Birla et al., "Time-Overcurrent Relay Coordination: A Review". *Int. J. Emerging Elect. Power Syst.*,vol.2, no. 2, 2005.

[11]. M.H. Hussain et al., "Optimal Overcurrent Relay Coordination: a Review." *Procedia Eng*,vol.53,, pp.332-336, 2013.

[12]. C. W So et al., “Application of Genetic Algorithm to Overcurrent Relay Grading Coordination," in *Proc. 4th Int. Conf. Advances Power System Control, Operation and Management,* Hong Kong.,1997, pp. 283-287.

[13]. F. Kavehnia et al, "Optimal coordination of directional over current relays in power system using genetic algorithm", in *Proc.41st Int. Universities Power Eng. Conf.,* 2006, pp. 824-827.

[14]. F. Razavi et al., “A new comprehensive genetic algorithm method for optimal overcurrent relays coordination", *Electric Power Syst. Research,* vol. 78, no. 4, pp. 713-720, Jun. 2008.

[15]. A. S. Noghabi et al., "Considering different network topologies in optimal overcurrent relay coordination using a hybrid GA", *IEEE Trans. Power Del,* vol. 24, no. 4, pp. 1857-1863, 2009.

[16]. P.P. Bedekar and S.R. Bhide, "Optimum coordination of overcurrent relay timing using continuous genetic algorithm", *Expert Syst. with Applicat.,* vol. 38, no. 9, pp. 11286-11292, 2011.

[17]. C.W. So and K.K.Li, "Overcurrent relay coordination by evolutionary programming", *Electric Power Syst. Research,* vol. 53, no. 2, pp. 83-90, 2000.

[18]. C.W. So and K.K. Li, "Intelligent method for protection coordination", in *Proc. 2004 IEEE Int. Conf. on Electric Utility Deregulation, Restructuring and Power Technologies*,2004,pp. 378-382.

[19]. R.Thangaraj et al., "Optimal coordination of over-current relays using modified differential evolution algorithms", *Eng. Applicat. of Artificial Intell.,* vol. 23, no. 5, pp. 820-829, Feb. 2010.

[20]. R.Thangaraj et al., "New mutation schemes for differential evolution algorithm and their application to the optimization of directional over-current relay settings", *Applied Math. and Computation,* vol. 216, no. 2, pp. 532-544, 2010.

[21]. Moirangthem et al., "Adaptive differential evolution algorithm for solving non-linear coordination problem of directional overcurrent relays", *IET Generation, Transmission and Distribution,* vol. 7, no. 4, pp. 329-336, 2013.

[22]. R.Thangaraj et al., "Overcurrent relay coordination by Differential Evolution algorithm", in *IEEE Int. Conf. on Power Electronics, Drives and Energy Syst.*, 2012, .

[23]. S.P.N Sheetekela, "Design of Power System Stabilizers using Evolutionary Algorithms," M.Sc Thesis, Dept. Elect.l Eng., UCT, Cape Town, 2010.

[24]. T.Mulumba, "Application of Differential Evolution to Power System Stabiliser Design," M.Sc Thesis, Dept. Elect.l Eng., UCT, Cape Town, 2010.

[25]. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning.* Reading, Mass, Addison-Wesley Pub. Co.,1989

[26]. A. P. Engelbrecht, *Computational Intelligence: An Introduction.* Chichester, England: J. Wiley & Sons, 2002.

[27]. H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm: I Continuous parameter optimization," *IEEE TransEvol.Comput*,vol.1, no.1, pp.25-49, 1993.

[28]. J. Greene, "The basic idea behind the Breeder Genetic Algorithm", Dept. Elect. Eng., Technical Report, UCT, 23 August 2005.

[29]. R. L. Hauptand S. E. Haupt, *Practical Genetic Algorithms*. Hoboken, NJ: Wiley-Interscience, 2004

[30]. J. L. Blackburn, *Protective Relaying: Principles and Applications.* New York, NY: Dekker, 1998.

[31]. L. G. Hewitson et al., *Practical Power System Protection.* Oxford: Elsevier/Newnes, 2004.

[32]. T. Bäck and H. P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization, "*IEEE Trans Evol. Comput.*, vol.1, no. 1, pp.1-23, 1993.

[33]. D. Beasly et al., "An overview of genetic algorithms: Part 1. Fundamentals, "*University Computing*,vol.15, no.2, pp. 58-69, 1993.

[34]. S. Baluja, "Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning," Carnigie Mellon University, Technical report CMU-CS-94-163, 1994.

[35]. A. K. Jain et al., "Artificial neural networks: A tutorial", *IEEE Trans. Comput.,* vol. 29, no. 3, pp. 31-44.

[36]. T. Kohonen, "The Self-organizing Map", *Proc. of the IEEE*, Vol. 78, No. 9, 1464-1480, 1990.

[37]. H. Demuthand M. Beale. *Neural Network Toolbox for Use with MATLAB: User's Guide*. Natick, Mass: MathWorks, Inc, 1998.

[38]. K.A. Folly and G.K. Venayagamoorthy, "Effects of Learning Rate on the Performance of the Population Based Incremental Learning Algorithm," in *Neural Networks, Int. Joint Conf. On Neural Networks*, vol., no., pp.861- 868, 2009.

[39]. K.A Folly and G.K. Venayagamoorthy, "Optimal Tuning of System Stabilizer Parameters Using PBIL with Adaptive Learning Rate," *IEEE Power and Energy Society General Meeting,* vol., no., pp.1-6, 2010

[40]. C. A. Coello Coello, "Theoretical and Numerical Constraint-Handling Techniques used with Evolutionary Algorithms: a Survey of the State of the Art", *Computer Methods in Applied Mechanics and Engineering*, Vol. 191, No. 11–12, pp. 1245-1287, 2002.

[41]. 'Power system test case archive', available at: http://www.ee.washington.edu/research/pstca/rts/pg_tcarts.htm

APPENDIX A

POWER SYSTEM DATA

The following data was used for the IEEE 24 bus system to study the application of evolutionary algorithms for optimal relay coordination. Data for the 230kV network is shown below.

Generator data

	Busbar	MVA	X''d (p.u)
Generator 4	13	232	0.32
Generator 5A	15	14	0.32
Generator 5B	15	182	0.3
Generator 6	16	182	0.3
Generator 7	18	471	0.4
Generator 8	21	471	0.4
Generator 9	22	53	0.28
Generator 10A	23	182	0.3
Generator 10B	23	412	0.3

Line data

Branch	R (p.u)	X (p.u)	Rating (A)
11 – 13	0.0061	0.0476	500
11 – 14	0.0054	0.0418	500
12 – 13	0.0061	0.0476	500
12 – 23	0.0124	0.0966	500
13- 23	0.0111	0.0865	500

14 – 16	0.005	0.0389	500
15 – 16	0.0022	0.0173	500
15 – 21	0.0063	0.049	500
15 – 21	0.0063	0.049	500
15 – 24	0.0067	0.0519	500
16 – 17	0.0033	0.0259	500
16 – 19	0.003	0.0231	500
17 – 18	0.0018	0.0144	500
17 – 22	0.0135	0.1053	500
18 – 21	0.0033	0.0259	500
18 – 21	0.0033	0.0259	500
19 – 20	0.0051	0.0396	500
19 – 20	0.0051	0.0396	500
20– 23	0.0028	0.0216	500
20 – 23	0.0028	0.0216	500
21 – 22	0.0087	0.0678	500

APPENDIX B

FAULT STUDY RESULTS

The results of a fault study performed on the IEEE 24 bus system are given below. For each relay coordination pair, the fault current values were obtained by simulating a three phase fault in front of the main relay.

Coordination Pairs		Fault Currents (A)	
Main	Backup	Main	Backup
R1	R23	6111	2392
R2	R22	6844	3108
R3	R25	5818	1989
R4	R24	6730	2999
R5	R1	10077	2072
	R3		1925
R6	R2	3205	3205
R7	R29	7910	2197
	R30		2197
	R31		1096
R8	R28	10674	5175
	R30		2081
	R31		1067
R9	R28	10674	5175
	R29		2081
	R31		1067
R10	R28	11757	5160
	R29		2130

	R30		2130
R11	R6	11119	1880
	R7		4310
	R33		3173
R12	R6	11209	1879
	R7		4356
	R32		3215
R13	R11	6021	4702
	R35		1320
R14	R11	9580	4385
	R34		5195
R15	R13	10096	3761
	R37		2747
R16	R13	10096	3761
	R36		2747
R17	R12	7848	5555
	R39		2293
R18	R12	7848	5555
	R38		2293
R19	R17	7572	1927
	R18		1927
	R41		3719
R20	R17	7572	1927
	R18		1927
	R40		3719

R21	R8	11718	1917
	R9		1917
	R15		2219
	R16		2219
R22	R3	10277	1968
	R26		2046
R23	R27	4321	4321
R24	R1	10409	2109
	R26		2049
R25	R5	11795	1805
	R19		1489
	R20		1489
R26	R4	11529	1423
	R19		1508
	R20		1508
R27	R7	12291	4280
	R32		3162
	R33		3119
R28	R6	10073	1894
	R32		3210
	R33		3199
R29	R9	11532	1885
	R15		2261
	R16		2261
	R42		1617

R30	R8	11532	1885
	R15		2261
	R16		2261
	R42		1617
R31	NO PAIR	1662	NO PAIR
R32	R34	6864	5552
	R35		1314
R33	R38	4784	2392
	R39		2392
R34	R36	9319	2848
	R37		2848
R35	R21	5774	2601
R36	R8	11427	1988
	R9		1988
	R16		2226
	R42		1647
R37	R8	11427	1988
	R9		1988
	R15		2226
	R42		1647
R38	R18	9296	1820
	R40		3739
	R41		3739
R39	R17	9296	1820
	R40		3739

	R41		3739
R40	R4	12471	1533
	R5		1963
	R20		1478
R41	R4	12471	1533
	R5		1963
	R19		1478
R42	R14	4938	1699

APPENDIX C

RELAY COORDINATION MATLAB CODES

The following Matlab m-files where used to study the application of evolutionary algorithms for optimal relay coordination.

Objective function file

Ndabeni Stenane

```
% Rev 1.0

% 11/09/2012



function val = ga14bus_17_Nov_2013(sol)



% Possible Time Multiplier Soulutions of each relay

% Possible Time Multiplier Soulutions of each relay

% Possible Time Multiplier Soulutions of each relay there are 14 relays

   temp    = size(sol);

   row     = temp(1);

   col     = temp(2);

% Make sol discrete before evaluation- to ensure the values are discrete

% before evaluation

% for i = 1:row

%       for j = 1: col

%          if mod(xin(i,j),0.05) < 1e-6
```

```
%           dscrtxin(i,j) = xin(i,j);
%         else
%           dscrtxin(i,j)= (ceil(xin(i,j)/0.05))* 0.05;
%         end
%     end
% end
  TM = sol;
% Pick-up is chosen based on the rating of the line, Fault values are
% obtained from a Power Systems Software


  Ipu = [600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600];
  %Duplicate the Ipu to match the size of the position array
  Ipu = ones(size(sol,1),1)*Ipu;


% Ifault_main = [3726 5155 8365 22135];
  Ifault_main = [6111 6844 5818 6730 10077 3205 7910 10674 10674 11757 11119
11209 6021 9580 10096 10096 7848 7848 7572 7572 11718 10277 4321 10409 11795
11529 12291 10073 11532 11532 1662 6864 4784 9319 5774 11427 11427 9296 9296
124471 12471 4938];
  %Duplicate the Ifault_main to match the size of the position array
  Ifault_main = ones(size(sol,1),1)*Ifault_main;
  %Operating time of main relays
```

```
    t_main = (0.14.*TM)./(((Ifault_main./Ipu).^0.02)-1);

    %First term of the objective funtion

    f1 = sum(t_main,2);

%Obtain the time diffrence between the back relay time and main relay time

%(with the CTI)

%Relay pairs

% relay_main_backup

relay_main_backup = [1 23; 2 22; 3 25; 4 24; 5 1; 5 3; 6 2; 7 29; 7 30; 7 31; 8 28; 8 30; 8
31; 9 28; 9 29; 9 31; 10 28; 10 29; 10 30; 11 6; 11 7; 11 33; 12 6; 12 7;12 32; 13 11; 13 35;
14 11; 14 34; 15 13; 15 37; 16 13; 16 36; 17 12; 17 39; 18 12; 18 38; 19 17; 19 18; 19 41;
20 17; 20 18; 20 40; 21 8; 21 9; 21 15; 21 16; 22 3; 22 26; 23 27; 24 1; 24 26; 25 5; 25 19;
25 20; 26 4; 26 19; 26 20; 27 7; 27 32; 27 33; 28 6; 28 32; 28 33; 29 9; 29 15; 29 16; 29
42; 30 8; 30 15; 30 16; 30 42; 32 34; 32 35; 33 38; 33 39; 34 36; 34 37; 35 21; 36 8; 36 9;
36 16; 36 42; 37 8; 37 9; 37 15; 37 42; 38 18; 38 40; 38 41; 39 17; 39 40; 39 41; 40 4; 40
5; 40 20; 41 4; 41 5; 41 19; 42 14]';

Ifault_backup = [2392 3108 1989 2999 2072 1925 3205 2197 2197 1096 5175 2081 1067
5175 2081 1067 5160 2130 2130 1880 4310 3173 1879 4356 3215 4702 1310 4385 5195
3761 2747 3761 2747 5555 2293 5555 2293 1927 1927 3719 1927 1927 3719 1917 1917
2219 2219 1968 2046 4321 2109 2049 1805 1489 1489 1423 1508 1508 4280 3162 3119
1894 3210 3199 1885 2261 2261 1617 1885 2261 2261 1617 5552 1314 2392 2392 2848
2848 2601 1988 1988 2226 1647 1988 1988 2226 1647 1820 3739 3739 1820 3739 3739
1533 1963 1478 1533 1963 1478 1699];

%RELAY COORDINATION

%Coordination margin

    CTI             = 0.3;

%Coordination measure

% dt              = t_backup - t_main - CTI;
```

```
main_t   = [];
main_p   = [];
backup_t = [];
main_f   = [];
backup_f = [];
dt       = [];
pen_dt   = [];
main_backup = size (relay_main_backup);
main_backup_pair = main_backup(2);
for i = 1:row
   for j = 1:main_backup_pair
      main_f(i,j)   = Ifault_main(i,relay_main_backup(1,j));
      main_p(i,j)   = Ipu(i,relay_main_backup(1,j));
      main_t(i,j)   = 0.14*TM(i,relay_main_backup(1,j))/((main_f(i,j)/main_p(i,j))^0.02 -
1);
      backup_f(i,j) = Ifault_backup(j);
      backup_p(i,j) = Ipu(i,relay_main_backup(2,j));
      backup_t(i,j)                                                                    =
(0.14*(TM(i,relay_main_backup(2,j))))/((backup_f(i,j)/backup_p(i,j))^0.02 -1);

      dt(i,j)       = backup_t(i,j) - main_t(i,j)- CTI;
      if dt(i,j) < 0
         pen_dt(i,j) = main_t(i,j)+ CTI - backup_t(i,j);
```

```
            elseif dt(i,j)>=0
                pen_dt(i,j) = 0;
            end
        end
    end
%     f1 = sum(main_t,2);
    f2 = sum(dt,2);
    f3 = sum(backup_t,2);
    g1 = sum(pen_dt,2); %Think about this.
    %PENALTIES
    pen_lb   =   [];
    pen_ub   =   [];
    for i = 1:row
        for j = 1: size (TM,2)
            if TM(i,j)> 1.0
                pen_ub(i,j) = TM(i,j)-1;
            elseif TM(i,j)< 1.0
                pen_ub(i,j) = 0;
            end
            if TM(i,j) < 0.01
                pen_lb(i,j) = 0.01 - TM(i,j);
            elseif TM(i,j) > 0.01
```

```
            pen_lb(i,j) = 0;
        end
    end
end
g2 = sum(pen_lb,2);
g3 = sum(pen_ub,2);
% Objective Function
% Constants
W = 50;
w_tm       = 1.0;      % operating time of main relays
w_tb       = 0.0;
w_dt       = 0.0;
w_pen_dt    = 1;
w_pen_ub    = 1;
w_pen_lb    = 1;
%Objective function as shown in Electric Power Systems Research 78 (2008)
%713- 720)

%Note: Evaluate the penalty before putting it in the objective function
val = (w_tm* f1 + w_tb*f3 + w_dt*f2+ W*(w_pen_dt*max(0, g1)+ w_pen_lb*max(0, g2) + w_pen_ub*max(0, g3))); % Ndabeni
```

GA code

```
%GENETIC ALGORITHM

% minimizes the objective function designated in ff

% Before beginning, set all the parameters in parts

% I, II, and III

% Haupt & Haupt

% 2003

%_____________________________________________________

clf

clc

clear

close all

mr_max            = 10;

miscoord          = [];

dt_run            = [];

fitrecrun         = [];

f1bestrecrun      = [];

f2bestrecrun      = [];

f3bestrecrun      = [];

mr_bestever       = inf;

good_dt           = 0;

good_ub           = 0;
```

```
good_lb           = 0;
good_dt_lb_ub     = 0;
TMbestrec         = [];
for mr = 1:mr_max
   % I Setup the GA
   % ff     ='ga14bus'; % objective function
   npar      =  42;                    % number of optimization variables
   varhi     =  1.0;
   varlo     =  0.01;                   % variable limits

   %_____________________________________________________________
   % II Stopping criteria
   maxit     =  1500;                   % max number of iterations
   mincost   =  -9999999;               % minimum cost
   %_____________________________________________________________
   % III GA parameters
   popsize   =  100;                    % set population size
   mutrate   =  0.02;                   % set mutation rate
   selection =  0.5;                    % fraction of population kept
   Nt        =  npar;                   % continuous parameter GA Nt=#variables
   keep      =  floor(selection*popsize);     % #population members that survive
   nmut      =  ceil((popsize-1)*Nt*mutrate);  % total number of mutations
```

```
M          =   ceil((popsize-keep)/2);            % number of matings

%_____________________________________________________________

% Create the initial population

iga                        = 0;              % generation counter

par                        = (varhi-varlo)*rand(popsize,npar)+varlo; % random

% Make the variables discrete before evaluation

temp_par                    =   size(par);

row_par                     =   temp_par(1);

col_par                     =   temp_par(2);

for i = 1:row_par

   for j = 1: col_par

      if mod(par(i,j),0.01) < 1e-10

         par(i,j)          = par(i,j);

      else

         par(i,j)          = (ceil(par(i,j)/0.01))* 0.01;

      end

   end

end

cost         = ga14bus_17_Nov_2013(par);                % calculates population cost using
objective function

[cost,ind]  = sort(cost);                    % min cost in element 1

par         = par(ind,:);                    % sort continuous

minc(1)     = min(cost);                     % minc contains min of population cost
```

```
meanc(1)    = mean(cost);                    % meanc contains mean of population cost

fitrec      = [];

%_____________________________________________________________

% Iterate through generations

while iga<maxit

   iga     = iga+1; % increments generation counter

   %_____________________________________________________________

   % Pair and mate

   M       = ceil((popsize-keep)/2);              % number of matings

   prob    = flipud([1:keep]'/sum([1:keep]));     % weights chromosomes

   odds    = [0 cumsum(prob(1:keep))'];           % probability distribution function

   pick1   = rand(1,M);                           % mate #1

   pick2   = rand(1,M);                           % mate #2


   % ma and pa contain the indicies of the chromosomes that will mate

   ic = 1;

   while ic<= M

      for id = 2:keep+1

         if pick1(ic)<=odds(id) & pick1(ic)>odds(id-1)

            ma(ic)            =id-1;

         end

         if pick2(ic)<=odds(id) & pick2(ic)>odds(id-1)
```

```
            pa(ic)          =id-1;
        end
    end
    ic = ic+1;
end
%_____________________________________________________
% Performs mating using single point crossover
ix = 1:2:keep; % index of mate #1
xp = ceil(rand(1,M)*Nt); % crossover point
r  = rand(1,M); % mixing parameter
for ic=1:M
    xy                  =  par(ma(ic),xp(ic))-par(pa(ic),xp(ic));  % ma and pa mate
    par(keep+ix(ic),:)      =  par(ma(ic),:);                   % 1st offspring
    par(keep+ix(ic)+1,:)    =  par(pa(ic),:);                   % 2nd offspring
    par(keep+ix(ic),xp(ic))   =  par(ma(ic),xp(ic))-r(ic).*xy;        % 1st
    par(keep+ix(ic)+1,xp(ic)) =  par(pa(ic),xp(ic))+r(ic).*xy;        % 2nd

    if xp(ic)<npar                                          % crossover when last variable
not selected
        par(keep+ix(ic),:)                  =           [par(keep+ix(ic),1:xp(ic))
par(keep+ix(ic)+1,xp(ic)+1:npar)];
        par(keep+ix(ic)+1,:)                =           [par(keep+ix(ic)+1,1:xp(ic))
par(keep+ix(ic),xp(ic)+1:npar)];
```

```
   end % if
end
%_____________________________________________________
% Mutate the population
mrow                    =   sort(ceil(rand(1,nmut)*(popsize-1))+1);
mcol                    =   ceil(rand(1,nmut)*Nt);

for ii=1:nmut
   par(mrow(ii),mcol(ii))     =   (varhi-varlo)*rand + varlo;
   % mutation
end % ii
%_____________________________________________________
% The new offspring and mutated chromosomes are evaluated
% Make the variables discrete before evaluation
for i = 1:row_par
   for j = 1: col_par
      if mod(par(i,j),0.01) < 1e-10
         par(i,j)          = par(i,j);
      else
         par(i,j)          = (ceil(par(i,j)/0.01))* 0.01;
      end
   end
```

```
    end
    cost                    =   ga14bus_17_Nov_2013(par);
    %_____________________________________________________
    % Sort the costs and associated parameters
    [cost,ind]              =   sort(cost);
    par                     =   par(ind,:);
    %_____________________________________________________
    % Do statistics for a single nonaveraging run
    minc(iga+1)             =  min(cost);
    meanc(iga+1)            =  mean(cost);
    %_____________________________________________________
    % Stopping criteria
    if iga>maxit | cost(1)<mincost
       break
    end
    [iga cost(1) mr]
    fitrec                  = [fitrec,cost(1)];
    plot(fitrec)
    drawnow
    bestever                = cost(1);
end %iga
%_____________________________________________________
```

```
xx = par(1,:);

TM_1best    = xx (1);

TM_2best    = xx (2);

TM_3best    = xx (3);

TM_4best    = xx (4);

TM_5best    = xx (5);

TM_6best    = xx (6);

TM_7best    = xx (7);

TM_8best    = xx (8);

TM_9best    = xx (9);

TM_10best   = xx (10);

TM_11best   = xx (11);

TM_12best   = xx (12);

TM_13best   = xx (13);

TM_14best   = xx (14);

TM_15best   = xx (15);

TM_16best   = xx (16);

TM_17best   = xx (17);

TM_18best   = xx (18);

TM_19best   = xx (19);

TM_20best   = xx (20);

TM_21best   = xx (21);
```

TM_22best = xx (22);

TM_23best = xx (23);

TM_24best = xx (24);

TM_25best = xx (25);

TM_26best = xx (26);

TM_27best = xx (27);

TM_28best = xx (28);

TM_29best = xx (29);

TM_30best = xx (30);

TM_31best = xx (31);

TM_32best = xx (32);

TM_33best = xx (33);

TM_34best = xx (34);

TM_35best = xx (35);

TM_36best = xx (36);

TM_37best = xx (37);

TM_38best = xx (38);

TM_39best = xx (39);

TM_40best = xx (40);

TM_41best = xx (41);

TM_42best = xx (42);

```
TMbest = [TM_1best TM_2best TM_3best TM_4best TM_5best TM_6best TM_7best TM_8best TM_9best TM_10best TM_11best TM_12best TM_13best TM_14best TM_15best TM_16best TM_17best TM_18best TM_19best TM_20best TM_21best TM_22best TM_23best TM_24best TM_25best TM_26best TM_27best TM_28best TM_29best TM_30best TM_31best TM_32best TM_33best TM_34best TM_35best TM_36best TM_37best TM_38best TM_39best TM_40best TM_41best TM_42best] %TM_43 TM_44 TM_45 TM_46 TM_47 TM_48 TM_49 TM_50 TM_51 TM_52 TM_53 TM_54 TM_55 TM_56 TM_57 TM_58 TM_59 TM_60 TM_61 TM_62 TM_63 TM_64 TM_65 TM_66 TM_67 TM_68 ];

pen_ub_best = [];

pen_lb_best = [];

for i = 1: size (TMbest,2)

    if TMbest(i)> 1.0

        pen_ub_best(i) = TMbest(i)-1.0;

    elseif TMbest(i)<= 1.0

        pen_ub_best(i) = 0;

    end

    if TMbest(i) < 0.01

        pen_lb_best(i) = 0.01 - TMbest(i);

    elseif TMbest(i) >= 0.01

        pen_lb_best(i) = 0;

    end

end

% Pick-up is chosen based on the rating of the line, Fault values are

% obtained from a Power Systems Software
```

```
Ipu = [600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600];

%Duplicate the Ipu to match the size of the position array

Ipu = ones(size(TMbest,1),1)*Ipu;

% Ifault_main = [3726 5155 8365 22135];

Ifault_main = [6111 6844 5818 6730 10077 3205 7910 10674 10674 11757 11119
11209 6021 9580 10096 10096 7848 7848 7572 7572 11718 10277 4321 10409 11795
11529 12291 10073 11532 11532 1662 6864 4784 9319 5774 11427 11427 9296 9296
124471 12471 4938];

%Duplicate the Ifault_main to match the size of the position array

Ifault_main = ones(size(xx,1),1)*Ifault_main;

%Operating time of main relays

t_main_best = (0.14.*TMbest)./(((Ifault_main./Ipu).^0.02)-1);

%First term of the objective funtion

f1best = sum(t_main_best,2);

%Obtain the time diffrence between the back relay time and main relay time

%(with the CTI)

%Relay pairs

% relay_main_backup

relay_main_backup = [1 23; 2 22; 3 25; 4 24; 5 1; 5 3; 6 2; 7 29; 7 30; 7 31; 8 28; 8 30;
8 31; 9 28; 9 29; 9 31; 10 28; 10 29; 10 30; 11 6; 11 7; 11 33; 12 6; 12 7;12 32; 13 11; 13
35; 14 11; 14 34; 15 13; 15 37; 16 13; 16 36; 17 12; 17 39; 18 12; 18 38; 19 17; 19 18; 19
41; 20 17; 20 18; 20 40; 21 8; 21 9; 21 15; 21 16; 22 3; 22 26; 23 27; 24 1; 24 26; 25 5; 25
```

19; 25 20; 26 4; 26 19; 26 20; 27 7; 27 32; 27 33; 28 6; 28 32; 28 33; 29 9; 29 15; 29 16; 29 42; 30 8; 30 15; 30 16; 30 42; 32 34; 32 35; 33 38; 33 39; 34 36; 34 37; 35 21; 36 8; 36 9; 36 16; 36 42; 37 8; 37 9; 37 15; 37 42; 38 18; 38 40; 38 41; 39 17; 39 40; 39 41; 40 4; 40 5; 40 20; 41 4; 41 5; 41 19; 42 14]';

Ifault_backup = [2392 3108 1989 2999 2072 1925 3205 2197 2197 1096 5175 2081 1067 5175 2081 1067 5160 2130 2130 1880 4310 3173 1879 4356 3215 4702 1310 4385 5195 3761 2747 3761 2747 5555 2293 5555 2293 1927 1927 3719 1927 1927 3719 1917 1917 2219 2219 1968 2046 4321 2109 2049 1805 1489 1489 1423 1508 1508 4280 3162 3119 1894 3210 3199 1885 2261 2261 1617 1885 2261 2261 1617 5552 1314 2392 2392 2848 2848 2601 1988 1988 2226 1647 1988 1988 2226 1647 1820 3739 3739 1820 3739 3739 1533 1963 1478 1533 1963 1478 1699];

%RELAY COORDINATION

%Coordination margin

CTI = 0.3;

%Coordination measure

% dt = t_backup - t_main - CTI;

main_t_best = [];

main_p = [];

backup_t_best = [];

main_f = [];

backup_f = [];

dt_best = [];

pen_dt_best = [];

main_backup = size (relay_main_backup);

main_backup_pair = main_backup(2);

```
for i = 1:size(xx,1)
    for j = 1:main_backup_pair
        main_f(i,j)        = Ifault_main(i,relay_main_backup(1,j));
        main_p(i,j)        = Ipu(i,relay_main_backup(1,j));
        main_t_best(i,j)                                                    =
0.14*TMbest(i,relay_main_backup(1,j))/((main_f(i,j)/main_p(i,j))^0.02 -1);
        backup_f(i,j)      = Ifault_backup(j);
        backup_p(i,j)      = Ipu(i,relay_main_backup(2,j));
        backup_t_best(i,j)                                                  =
(0.14*(TMbest(i,relay_main_backup(2,j))))/((backup_f(i,j)/backup_p(i,j))^0.02 -1);


        dt_best(i,j)       = backup_t_best(i,j) - main_t_best(i,j)- CTI;
        if dt_best(i,j) < 0
           pen_dt_best(i,j)    = main_t_best(i,j)+ CTI - backup_t_best(i,j);
        elseif dt_best(i,j)>=0
           pen_dt_best(i,j)    = 0;
        end
    end
end
dt_gen     = dt_best;
dt_run     = [dt_run;dt_gen];
miscoordgen = sum(dt_best< 0.0);
miscoord = [miscoord; miscoordgen];
```

```
f2best       = sum(dt_best,2);

f3best       = sum(backup_t_best,2);

g1best       = sum(pen_dt_best,2);

g2best       = sum(pen_lb_best,2);

g3best       = sum(pen_ub_best,2);

if sum(dt_best<0)==0

   good_dt = good_dt + 1;

end

if sum(TMbest < 0.01) == 0

   good_lb = good_lb + 1;

end

if sum(TMbest > 1.0) == 0

   good_ub = good_ub + 1;

end

if  sum(TMbest > 1.000000)==0 & sum(dt_best<0.001) == 0 & sum(TMbest <0.01)==0

   good_dt_lb_ub = good_dt_lb_ub + 1;

end

fitrecrun          = [fitrecrun; fitrec];

f1bestrecrun       = [f1bestrecrun;f1best]; %A record of main operating times

f2bestrecrun       = [f2bestrecrun;f2best]; %A record of grading margins

f3bestrecrun       = [f3bestrecrun;f3best]; %A record of backup operating times

mr
```

```
   TMbestrec = [TMbestrec;TMbest];
   if bestever < mr_bestever
      if sum(TMbest > 1.000000)==0 & sum(dt_best<-0.001) == 0 & sum(TMbest <0.01)==0
         mr_bestever     = bestever;
         mr_f1_bestever  = f1best;
         mr_f2_bestever  = f2best;
         mr_f3_bestever  = f3best;
         mr_TMbestever   = TMbest;
         fitrecbest      = fitrec;
      end
   end
end
fitrecrun_fin          = fitrecrun(:,maxit);
[fitrecrun_sort,ind_fitrecbest]       = sort(fitrecrun_fin);
% fitrecbest             = fitrecrun(ind_fitrecbest(1),:);
fitrecrunave           = mean(fitrecrun,1);
fitrecrunstd           = std(fitrecrun(:,maxit),1,1);
f1bestrecrunave        = mean(f1bestrecrun,1);
f1bestrecrunstd        = std(f1bestrecrun,1,1);
f2bestrecrunave        = mean(f2bestrecrun,1);
f2bestrecrunstd        = std(f2bestrecrun,1,1);
f3bestrecrunave        = mean(f3bestrecrun,1);
```

```
f3bestrecrunstd        = std(f3bestrecrun,1,1);

successrate            = (good_dt_lb_ub/mr)*100%

RESULTS

plot(fitrecrunave)

title('Average fitness')

figure

plot(fitrecbest)

title('Best fitness')
```

BGA code

```
% AMBA   Breeder genetic Algorithm with adaptive mutation

%

% This version of AMBA passes to FUNC  a NVARS x POP matrix of random

% numbers in the range 0-1.  Each column vector should be interpreted

% as a trial solution to the optimisation task in hand. The evaluating

% function should return a column vector of fitness values, which AMBA will

% try to maximise.

clc

clf

clear all

mr_max             = 10;

fitrecrun          = [];

f1bestrecrun       = [];
```

```
f2bestrecrun       = [];
f3bestrecrun       = [];
miscoord           = [];
good_dt            = 0;
good_ub            = 0;
good_lb            = 0;
good_dt_lb_ub      = 0;
mr_best            = inf;
TMbestrec          =[];
verybest = [];
for mr = 1: mr_max
   maxgen    = 1500;                % number of generations
   pop       = 100;                 % population size (no. trial solutions)
   nvars     = 42 ;                 % number of variables
   l_bound   = 0.01;
   u_bound   = 1.0;
   res       = 0.01;
   thr =round(pop*10/100);          % 10%, 30%, 50% selection threshold

   fitrec = [];

   % INITIALIZE THE MUTATION RATE
```

```
delta = 0.1;                            % nominal mutation rate
% INITIALIZE THE POPULATION
T = l_bound + (u_bound -l_bound).*rand(nvars,pop); % initial matrix of trial solutions
for gen = 1:maxgen
   %Make T discrete
   for j = 1:pop
      for i = 1: nvars
         if mod(T(i,j),res) < 1e-10
            dscrtT(i,j) = T(i,j);
         else
            dscrtT(i,j)= (ceil(T(i,j)/res))* res;
         end
      end
   end
   T      = dscrtT;

   %EVALUATE THE FITNESS OF THE INDIVIDUALS
   f      = fu14bus_17_November_2013(T);
   f      = f';                        % row vector of fitnesses
   flow =  mean(f(2:3));               % mean fitness with lower mutation
   fhigh = mean(f(4:5));          % mean fitnes with higher mutation
   %ADAPTIVE MUTATION
```

```
if flow < fhigh                    % if lower mutation population improves fitness
   delta = 0.95*delta;             % decrease mutation rate
else
   delta = 1.05*delta;             % otherwise, increase it
end
%FIND BEST SOLUTION
[y,i]  = sort(f);                  % sort by fitness in ascending order
S      = T(:,i);                   % Sorted trial solutions
best    = fu14bus_17_November_2013(S(:,1));          % best fitness sofar in the population:
fitrec = [fitrec,best];            % record it for all the number of generations
%PERFORM SELECTION
pool = S(:,1:thr);                 % survivors (breeding pool based on the threshold)


T(:,1) = S(:,1);                   % elitist insertion
%PERFORM RECOMBINATION
c = 0;
b = 0;
for i =2:pop                       % construct rest of population
   R = randperm(thr);              % pick two different parents at random
   rnd = rand;
   if rnd < 0.1                                % discrete recombination
      mask = rand(nvars,1) > 0.5;
```

```
            T(:,i) = mask.*pool(R(1)) +(~mask).*pool(R(2));
        else
            rr = -0.25+ 1.5*rand(nvars,1);                    % volume recomb
            T(:,i) = rr.*pool(:,R(1)) + (1-rr).*pool(:,R(2));
        end

        % mutate lower/higher half of population at lower/higher rate
        r1 = 1+floor(nvars*rand);    % random integer in range (1,nvars)
        if i < 51
            T(r1,i) = T(r1,i)+delta*(randn/1.1);% Lower mutation
            c = c+1;
        else
            T(r1,i) = T(r1,i)+delta*(1.1*randn); %Higher mutation
            b = b+1;
        end
    end
    % PERFORM MUTATION
    %Divide the new population into two halves X(top half) and Y (lower half)
    % Apply 1/2*r to X LOWER MUTATION

%           T(:,1:0.5*pop) = T(:,1:0.5*pop) + delta*0.5;
%           X = T(:,1:0.5*pop);
```

```
%            X = X + delta*(randn/1.1);
%        %    Apply 2*r to Y
%        %
%            T(:,(0.5*pop+1):pop) = T(:,(0.5*pop+1):pop) + 2*delta;
%            Y = T(:,(0.5*pop+1):pop);
%            Y = Y + delta*(1.1*randn);
%        %    EVALUATE THE AVARAGE FITNESS OF THE HALVES
%            f_xmean = mean(fu14bus(X))
%            f_ymean = mean(fu14bus(Y))
%        %    IMPLEMENT ADAPTIVE MUTATION
%            if f_xmean > f_ymean
%               delta = 0.95*delta;
%            else
%               delta = 1.05*delta;
%            end

      disp([' run ','generation ','mutation rate ','current best'])
      disp([mr,gen,delta,best])
%        plot(fitrec)                % incremental plot of fitness
%        drawnow
   end
   xx= S(:,1);
```

```
TM_1best = xx (1);

TM_2best = xx (2);

TM_3best = xx (3);

TM_4best = xx (4);

TM_5best = xx (5);

TM_6best = xx (6);

TM_7best = xx (7);

TM_8best = xx (8);

TM_9best = xx (9);

TM_10best = xx (10);

TM_11best = xx (11);

TM_12best = xx (12);

TM_13best = xx (13);

TM_14best = xx (14);

TM_15best = xx (15);

TM_16best = xx (16);

TM_17best = xx (17);

TM_18best = xx (18);

TM_19best = xx (19);

TM_20best = xx (20);

TM_21best = xx (21);

TM_22best = xx (22);
```

```
TM_23best = xx (23);

TM_24best = xx (24);

TM_25best = xx (25);

TM_26best = xx (26);

TM_27best = xx (27);

TM_28best = xx (28);

TM_29best = xx (29);

TM_30best = xx (30);

TM_31best = xx (31);

TM_32best = xx (32);

TM_33best = xx (33);

TM_34best = xx (34);

TM_35best = xx (35);

TM_36best = xx (36);

TM_37best = xx (37);

TM_38best = xx (38);

TM_39best = xx (39);

TM_40best = xx (40);

TM_41best = xx (41);

TM_42best = xx (42);

%

TMbest = [TM_1best; TM_2best; TM_3best; TM_4best; TM_5best; TM_6best;
TM_7best; TM_8best; TM_9best; TM_10best; TM_11best; TM_12best; TM_13best;
```

```
TM_14best; TM_15best; TM_16best; TM_17best; TM_18best; TM_19best; TM_20best;
TM_21best; TM_22best; TM_23best; TM_24best; TM_25best; TM_26best; TM_27best;
TM_28best; TM_29best; TM_30best; TM_31best; TM_32best; TM_33best; TM_34best;
TM_35best; TM_36best; TM_37best; TM_38best; TM_39best; TM_40best;TM_41best;
TM_42best ]';% TM_41 TM_42 TM_43 TM_44 TM_45 TM_46 TM_47 TM_48 TM_49
TM_50 TM_51 TM_52 TM_53 TM_54 TM_55 TM_56 TM_57 TM_58 TM_59 TM_60
TM_61 TM_62 TM_63 TM_64 TM_65 TM_66 TM_67 TM_68 ];


    % Pick-up is chosen based on the rating of the line, Fault values are

    % obtained from a Power Systems Software


    Ipu = [600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600];

    %Duplicate the Ipu to match the size of the position array

    Ipu = ones(size(TMbest,1),1)*Ipu;

    % Ifault_main = [3726 5155 8365 22135];

    Ifault_main = [6111 6844 5818 6730 10077 3205 7910 10674 10674 11757 11119
11209 6021 9580 10096 10096 7848 7848 7572 7572 11718 10277 4321 10409 11795
11529 12291 10073 11532 11532 1662 6864 4784 9319 5774 11427 11427 9296 9296
124471 12471 4938];

    %Duplicate the Ifault_main to match the size of the position array

    Ifault_main= ones(size(TMbest,1),1)*Ifault_main;

    %Operating time of main relays

    t_main_best = (0.14.*(TMbest))./(((Ifault_main./Ipu).^0.02)-1);

    %First term of the objective funtion
```

```
%     f1best = sum(t_main,2);

   %Obtain the time difference btween the back up relay time and main time

   %(with the CTI

   %Relay pairs

   relay_main_backup = [1 23; 2 22; 3 25; 4 24; 5 1; 5 3; 6 2; 7 29; 7 30; 7 31; 8 28; 8 30;
8 31; 9 28; 9 29; 9 31; 10 28; 10 29; 10 30; 11 6; 11 7; 11 33; 12 6; 12 7;12 32; 13 11; 13
35; 14 11; 14 34; 15 13; 15 37; 16 13; 16 36; 17 12; 17 39; 18 12; 18 38; 19 17; 19 18; 19
41; 20 17; 20 18; 20 40; 21 8; 21 9; 21 15; 21 16; 22 3; 22 26; 23 27; 24 1; 24 26; 25 5; 25
19; 25 20; 26 4; 26 19; 26 20; 27 7; 27 32; 27 33; 28 6; 28 32; 28 33; 29 9; 29 15; 29 16;
29 42; 30 8; 30 15; 30 16; 30 42; 32 34; 32 35; 33 38; 33 39; 34 36; 34 37; 35 21; 36 8; 36
9; 36 16; 36 42; 37 8; 37 9; 37 15; 37 42; 38 18; 38 40; 38 41; 39 17; 39 40; 39 41; 40 4;
40 5; 40 20; 41 4; 41 5; 41 19; 42 14]';

   % Ifault_backup

   Ifault_backup = [2392 3108 1989 2999 2072 1925 3205 2197 2197 1096 5175 2081
1067 5175 2081 1067 5160 2130 2130 1880 4310 3173 1879 4356 3215 4702 1310 4385
5195 3761 2747 3761 2747 5555 2293 5555 2293 1927 1927 3719 1927 1927 3719 1917
1917 2219 2219 1968 2046 4321 2109 2049 1805 1489 1489 1423 1508 1508 4280 3162
3119 1894 3210 3199 1885 2261 2261 1617 1885 2261 2261 1617 5552 1314 2392 2392
2848 2848 2601 1988 1988 2226 1647 1988 1988 2226 1647 1820 3739 3739 1820 3739
3739 1533 1963 1478 1533 1963 1478 1699];

   %Duplicate the Ifault_backup to match the size of the position array

   % Ifault_backup_ps = ones(size(sol,1),1)*Ifault_backup;



   %RELAY COORDINATION

   %Coordination margin

   CTI              = 0.3;

   temp_TMbest      = size(TMbest);
```

```
row_TMbest          = temp_TMbest(1);

col_TMbest          = temp_TMbest(2);

%Coordination measure

% dt                = t_backup - t_main - CTI;

main_t_best     = [];

main_p          = [];

backup_t_best   = [];

main_f          = [];

backup_f        = [];

dt_best         = [];

pen_dt_best     = [];

main_backup = size (relay_main_backup);

main_backup_pair = main_backup(2);

for i = 1:row_TMbest

    for j = 1:main_backup_pair

        main_f(i,j)         = Ifault_main(i,relay_main_backup(1,j));

        main_p(i,j)         = Ipu(i,relay_main_backup(1,j));

        main_t_best(i,j)                                                    =
0.14*TMbest(i,relay_main_backup(1,j))/((main_f(i,j)/main_p(i,j))^0.02 -1);

        backup_f(i,j)       = Ifault_backup(j);

        backup_p(i,j)       = Ipu(i,relay_main_backup(2,j));

        backup_t_best(i,j)                                                  =
(0.14*(TMbest(i,relay_main_backup(2,j))))/((backup_f(i,j)/backup_p(i,j))^0.02 -1);
```

```
        dt_best(i,j)        = backup_t_best(i,j) - main_t_best(i,j)- CTI;
        if dt_best(i,j) < -0.00100
            pen_dt_best(i,j) = main_t_best(i,j)+ CTI - backup_t_best(i,j);
            dt_best(i,j) = 0.0000
        elseif dt_best(i,j)>=0
            pen_dt_best(i,j) = 0;
        end
    end
end
pen_lb_best   =   [];
pen_ub_best   =   [];
for i = 1:row_TMbest
    for j = 1: col_TMbest
        if TMbest(i,j)> 1.0
            pen_ub_best(i,j) = TMbest(i,j)-1.0;
        elseif TMbest(i,j)<= 1.0
            pen_ub_best(i,j) = 0;
        end
        if TMbest(i,j) < 0.01
            pen_lb_best(i,j) = 0.01 - TMbest(i,j);
        elseif TMbest(i,j) >= 0.01
            pen_lb_best(i,j) = 0;
```

```
        end
    end
end
% Count the number of miscoordinations
coord_pair  = size (dt_best);
coord_pair  = coord_pair (2);
miscoord1   = (sum(dt_best<-0.001));
miscoord    = [miscoord; miscoord1];
f1best      = sum(t_main_best,2)
f2best      = sum(dt_best,2);
f3best      = sum(backup_t_best,2)
g1best      = sum(pen_dt_best,2);
g2best      = sum(pen_lb_best,2);
g3best      = sum(pen_ub_best,2);
if  sum(dt_best<0.00000) == 0
    good_dt = good_dt + 1;
end

if sum(TMbest < 0.010000) == 0
   good_lb = good_lb + 1;
end
if sum(TMbest > 1.0000) == 0
```

```
      good_ub = good_ub + 1;
   end
   if     sum(TMbest  >  1.0000)==0  &  sum(dt_best<-0.00100)  ==  0  &  sum(TMbest
<0.010000)==0
      good_dt_lb_ub = good_dt_lb_ub + 1;
   end

   fitrecrun          = [fitrecrun; fitrec];
   f1bestrecrun       = [f1bestrecrun;f1best]; %A record of main operating times
   f2bestrecrun       = [f2bestrecrun;f2best]; %A record of grading margins
   f3bestrecrun       = [f3bestrecrun;f3best]; %A record of backup operating times
   if best < mr_best
      if    sum(TMbest  >  1.0000)==0  &  sum(dt_best  <  -0.001)  ==  0  &  sum(TMbest
<0.01)==0
         mr_best     = best;
         mr_f1_best  = f1best;
         mr_f2_best  = f2best;
         mr_f3_best  = f3best;
         mr_TMbest   = TMbest;
         fitrecbest  = fitrec;
      end
   end
%     fitrecrun          = [fitrecrun; bestfitness];
```

```
    mr
    TMbestrec          = [TMbestrec;TMbest];
    verybest           = [verybest,mr_best];
end
fitrecrun_fin          = fitrecrun(:,maxgen);
fitrecrunave           = mean(fitrecrun,1);
fitrecrunstd           = std(fitrecrun(:,maxgen),1,1);
f1bestrecrunave        = mean(f1bestrecrun,1);
f1bestrecrunstd        = std(f1bestrecrun,1,1);
f2bestrecrunave        = mean(f2bestrecrun,1);
f2bestrecrunstd        = std(f2bestrecrun,1,1);
f3bestrecrunave        = mean(f3bestrecrun,1);
f3bestrecrunstd        = std(f3bestrecrun,1,1);
successrate            = (good_dt_lb_ub/mr)*100
%RESULTS
plot(fitrecrunave)
title('Average fitness')
figure
plot(fitrecbest)
title('Best fitness')
```

PBIL code

```
% PBIL
```

% PBIL generates random bitstrings, with control over the probability that a bit in a particular

% position is a '1'. This control is effected by a probality vector, which contains the bit-probabilities

% (numbers in the range 0-1).

% To generate a randomised bitstring with the distribution specified by the PV, the latter

% is 'sampled' by generating a uniformly random vector and comparing it element-by-element

% with the PV. Wherever an element of the PV is greater than the corresponding random

% element, a '1' is generated (otherwise a 0).

%To find a solution to an optimisation problem:

% 1. Initialise elements of the PV to 0.5 (ensuring uniformly-random bitstrings)

% Generate a population of uniformly-random bitstrings

% Interpret each bitstring as a solution to the problem and evaluate its merit

% in order to identify the "Best".

%2. Repeat the following:

%

% Adjust PV to favour the generation of bitstrings which resembe "Best"

% (slightly increase PV(i) if Best(i) =1 and decrease PV(i) if Best(i) =0)

```
% Generate a new population reflecting the modified distribution

% Interpret and evaluate each bitstring to find the new "Best"

% Until a satisfactory solution is found.


% Here is a very basic implementation, minimising the function f (fmax =0 at X=[1 2 3])

% NB: Using 7 bits per variable allows each variable to be expressed with 1/128

%(better than 1%) precision

clf

clc

close all

clear all

% hold on

echo off

TMbestrecrun    = [];

mr_max          = 10;

dt_run          = [];

miscoord        = [];

fitrecrun       = [];

f1bestrecrun    = [];

f2bestrecrun    = [];

f3bestrecrun    = [];

mr_bestever         = inf;
```

```
good_dt            = 0;
good_ub            = 0;
good_lb            = 0;
good_dt_lb_ub      = 0;
for mr = 1:mr_max
   % null vector to hold record of fitness
   bestfitness       = [];
   x1                = [];
   xx                = x1;
   adaptive          = 0;
   purelyadaptive    = 0;
   fixedtoadap       = 0;
   g_fixedtoadapt    = 0;
   adaptofixed       = 0;
   g_adaptofixed     = 100;
   LRmax             = 1.0;    %learning rate
   NLR               = (1- LRmax);
   n_LR              = 1.0;
   FF                = 0.005;   %forgetting factor
   pMutate           = 0.002;     % probability vector mutation rate
   shiftMutate       = 0.05;    % probability vector mutation shift magnitude
   bestgen           = inf;
```

```
bitstring          = 15;

bw                 = 2.^((bitstring-1):-1:0);  % vector of descending powers of 2 ('bitweights) each variable is made of 15 bits

nvars              = 42;

popsize            = 100;

maxgen             = 600;


updatefrom         = 2;


% initialise 45-element PV (15 bits per variable)

%INITIALIZE PV

PV          = 0.5*ones(1,nvars*bitstring);          % the size of the probability vector is number of variable x bitstrings of each variable

fitrec = [];

%GENERATE A RANDOM INITIAL SOLUTION BASED ON PV

count = 0;

for g = 1:maxgen                                    % will run for 250 generations

   if adaptive == 1    %Adaptive LR

      if purelyadaptive

         LR = g/maxgen*LRmax;

      elseif fixedtoadap

         if g < g_fixedtoadapt

            LR = LRmax;
```

```
            else
               LR = g/maxgen*LRmax;
            end
         elseif adaptofixed
            if g < g_adaptofixed
               LR = g/g_adaptofixed*LRmax;
            else
               LR = LRmax;
            end
         end
      elseif  adaptive == 0        %Non-Adaptive LR
         LR = LRmax;
      end



      %Generate a new population reflecting the modified distribution
      for pop_ind                  =1:popsize       % t trial solution (population size -30) based on PV (Population)


         ts(pop_ind,:)             = rand(size(PV)) < PV;              % generate bitstring (if true value =1; otherwise=0)


         xmin                      = 0.01;
```

```
        xmax                    = 1.0;

        xdec(pop_ind,:)                                                 =
(reshape(ts(pop_ind,:),nvars,bitstring)*bw'/2^(bitstring))';        % 14 variables of 15 bits
each

        rng                     = xmax - xmin;

        xcont(pop_ind,:)            = xmin + rng*xdec(pop_ind,:);

        temp_xcont                  =   size(xcont);

        row_xcont                   =   temp_xcont(1);

        col_xcont                   =   temp_xcont(2);


        for i = 1:row_xcont
           for j = 1: col_xcont
              if mod(xcont(i,j),0.01) < 1e-10
                 xdiscr(i,j)        = xcont(i,j);
              else
                 xdiscr(i,j)        = (ceil(xcont(i,j)/0.01))* 0.01;
              end
           end %end of columns of discrete
        end% end of rowns of discrete
     end %end of trial solutions
     %       cost        = pbil14bus_17_Nov_2013_2(xdiscr);      % calculates population
cost using objective function

     cost        = pbil14bus_17_Nov_2013_2(xdiscr);      % calculates population cost
using objective function (Evaluate solutions)
```

```
[costsort,ind]  = sort(cost);                    % min cost in element 1
tssortpop       = ts(ind,:);                     % sort ts
xdiscrsortpop   = xdiscr(ind,:);
bestsolpop      = tssortpop(1,:);
worstsolpop     = tssortpop(popsize,:);
bestpop         = costsort(1)




if bestpop < bestgen
   bestgen = bestpop;
   bestsolgen = bestsolpop;
   xdiscrsortgen = xdiscrsortpop(1,:);
   count = count + 1;
end
PV = (1-LR)*PV + LR*bestsolpop;          % update the probability vector
PV = PV - FF*(PV - 0.5);


%        PV = (1 - LR)*PV + LR*bestsolever;


%               Mutate the Probability vector
%        for i = 1 : size(PV)
```

```
%            if rand < pMutate
%               if PV(i) > 0.5
%                  PV(i) = PV(i)*(1- shiftMutate);
%               elseif PV(i) == 0.5
%                  PV(i) = PV(i);
%               elseif PV(i) < 0.5
%                  PV(i) = PV(i)*(1- shiftMutate) + shiftMutate;
%               end
%            end
%        end

   fitrec = [fitrec,bestgen];              % append the best fitness to fitness record
   [g bestgen mr]
   plot(fitrec)
   drawnow

   %        bestsol = bestsolgen;
   %        bestever                = cost(1);
end%end of generations
%     plot (fitrec)
%     figure
xx = xdiscrsortgen;
```

```
TM_1best    = xx (1);

TM_2best    = xx (2);

TM_3best    = xx (3);

TM_4best    = xx (4);

TM_5best    = xx (5);

TM_6best    = xx (6);

TM_7best    = xx (7);

TM_8best    = xx (8);

TM_9best    = xx (9);

TM_10best   = xx (10);

TM_11best   = xx (11);

TM_12best   = xx (12);

TM_13best   = xx (13);

TM_14best   = xx (14);

TM_15best   = xx (15);

TM_16best   = xx (16);

TM_17best   = xx (17);

TM_18best   = xx (18);

TM_19best   = xx (19);

TM_20best   = xx (20);

TM_21best   = xx (21);

TM_22best   = xx (22);
```

TM_23best = xx (23);

TM_24best = xx (24);

TM_25best = xx (25);

TM_26best = xx (26);

TM_27best = xx (27);

TM_28best = xx (28);

TM_29best = xx (29);

TM_30best = xx (30);

TM_31best = xx (31);

TM_32best = xx (32);

TM_33best = xx (33);

TM_34best = xx (34);

TM_35best = xx (35);

TM_36best = xx (36);

TM_37best = xx (37);

TM_38best = xx (38);

TM_39best = xx (39);

TM_40best = xx (40);

TM_41best = xx (41);

TM_42best = xx (42);

TMbest = [TM_1best TM_2best TM_3best TM_4best TM_5best TM_6best TM_7best TM_8best TM_9best TM_10best TM_11best TM_12best TM_13best TM_14best

```
TM_15best TM_16best TM_17best TM_18best TM_19best TM_20best TM_21best
TM_22best TM_23best TM_24best TM_25best TM_26best TM_27best TM_28best
TM_29best TM_30best TM_31best TM_32best TM_33best TM_34best TM_35best
TM_36best TM_37best TM_38best TM_39best TM_40best TM_41best TM_42best]
%TM_43 TM_44 TM_45 TM_46 TM_47 TM_48 TM_49 TM_50 TM_51 TM_52 TM_53
TM_54 TM_55 TM_56 TM_57 TM_58 TM_59 TM_60 TM_61 TM_62 TM_63 TM_64
TM_65 TM_66 TM_67 TM_68 ];

   pen_ub_best   =   [];

   pen_lb_best   =   [];

   for i = 1: size (TMbest,2)

      if TMbest(i)> 1.0

         pen_ub_best(i) = TMbest(i)-1.0;

      elseif TMbest(i)<= 1.0

         pen_ub_best(i) = 0;

      end

      if TMbest(i) < 0.01

         pen_lb_best(i) = 0.01 - TMbest(i);

      elseif TMbest(i) >= 0.01

         pen_lb_best(i) = 0;

      end

   end

   % Pick-up is chosen based on the rating of the line, Fault values are

   % obtained from a Power Systems Software
```

```
Ipu = [600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600 600
600 600 600];

%Duplicate the Ipu to match the size of the position array

Ipu = ones(size(TMbest,1),1)*Ipu;

% Ifault_main = [3726 5155 8365 22135];

Ifault_main = [6111 6844 5818 6730 10077 3205 7910 10674 10674 11757 11119
11209 6021 9580 10096 10096 7848 7848 7572 7572 11718 10277 4321 10409 11795
11529 12291 10073 11532 11532 1662 6864 4784 9319 5774 11427 11427 9296 9296
124471 12471 4938];

%Duplicate the Ifault_main to match the size of the position array

Ifault_main = ones(size(xx,1),1)*Ifault_main;

%Operating time of main relays

t_main_best = (0.14.*TMbest)./(((Ifault_main./Ipu).^0.02)-1);

%First term of the objective funtion

f1best = sum(t_main_best,2);

%Obtain the time diffrence between the back relay time and main relay time

%(with the CTI)

%Relay pairs

% relay_main_backup

relay_main_backup = [1 23; 2 22; 3 25; 4 24; 5 1; 5 3; 6 2; 7 29; 7 30; 7 31; 8 28; 8 30;
8 31; 9 28; 9 29; 9 31; 10 28; 10 29; 10 30; 11 6; 11 7; 11 33; 12 6; 12 7;12 32; 13 11; 13
35; 14 11; 14 34; 15 13; 15 37; 16 13; 16 36; 17 12; 17 39; 18 12; 18 38; 19 17; 19 18; 19
41; 20 17; 20 18; 20 40; 21 8; 21 9; 21 15; 21 16; 22 3; 22 26; 23 27; 24 1; 24 26; 25 5; 25
19; 25 20; 26 4; 26 19; 26 20; 27 7; 27 32; 27 33; 28 6; 28 32; 28 33; 29 9; 29 15; 29 16;
29 42; 30 8; 30 15; 30 16; 30 42; 32 34; 32 35; 33 38; 33 39; 34 36; 34 37; 35 21; 36 8; 36
```

9; 36 16; 36 42; 37 8; 37 9; 37 15; 37 42; 38 18; 38 40; 38 41; 39 17; 39 40; 39 41; 40 4; 40 5; 40 20; 41 4; 41 5; 41 19; 42 14]';

Ifault_backup = [2392 3108 1989 2999 2072 1925 3205 2197 2197 1096 5175 2081 1067 5175 2081 1067 5160 2130 2130 1880 4310 3173 1879 4356 3215 4702 1310 4385 5195 3761 2747 3761 2747 5555 2293 5555 2293 1927 1927 3719 1927 1927 3719 1917 1917 2219 2219 1968 2046 4321 2109 2049 1805 1489 1489 1423 1508 1508 4280 3162 3119 1894 3210 3199 1885 2261 2261 1617 1885 2261 2261 1617 5552 1314 2392 2392 2848 2848 2601 1988 1988 2226 1647 1988 1988 2226 1647 1820 3739 3739 1820 3739 3739 1533 1963 1478 1533 1963 1478 1699];

%RELAY COORDINATION

%Coordination margin

CTI = 0.3;

%Coordination measure

% dt = t_backup - t_main - CTI;

main_t_best = [];

main_p = [];

backup_t_best = [];

main_f = [];

backup_f = [];

dt_best = [];

pen_dt_best = [];

main_backup = size (relay_main_backup);

main_backup_pair = main_backup(2);

for i = 1:size(xx,1)

for j = 1:main_backup_pair

```
        main_f(i,j)        = Ifault_main(i,relay_main_backup(1,j));

        main_p(i,j)        = Ipu(i,relay_main_backup(1,j));

        main_t_best(i,j)                                                    =
0.14*TMbest(i,relay_main_backup(1,j))/((main_f(i,j)/main_p(i,j))^0.02 -1);

        backup_f(i,j)      = Ifault_backup(j);

        backup_p(i,j)      = Ipu(i,relay_main_backup(2,j));

        backup_t_best(i,j)                                                  =
(0.14*(TMbest(i,relay_main_backup(2,j))))/((backup_f(i,j)/backup_p(i,j))^0.02 -1);



        dt_best(i,j)       = backup_t_best(i,j) - main_t_best(i,j)- CTI;

        if dt_best(i,j) < -0.001

           pen_dt_best(i,j)    = main_t_best(i,j)+ CTI - backup_t_best(i,j);

           dt_best(i,j) = 0.0

        elseif dt_best(i,j)>=0

           pen_dt_best(i,j)    = 0;

        end

     end

  end

  dt_gen     = dt_best;

  dt_run     = [dt_run;dt_gen];

  miscoordgen = sum(dt_best<-0.001);

  miscoord = [miscoord; miscoordgen];

  f2best      = sum(dt_best,2);
```

```
f3best       = sum(backup_t_best,2);

g1best       = sum(pen_dt_best,2);

g2best       = sum(pen_lb_best,2);

g3best       = sum(pen_ub_best,2);

if sum(dt_best<-0.001)==0

   good_dt = good_dt + 1;

end

if sum(TMbest < 0.01) == 0

   good_lb = good_lb + 1;

end

if sum(TMbest > 1.0) == 0

   good_ub = good_ub + 1;

end

if    sum(TMbest  >  1.000000)==0  &  sum(dt_best<-0.001)  ==  0  &  sum(TMbest
<0.01)==0

   good_dt_lb_ub = good_dt_lb_ub + 1;

end

fitrecrun          = [fitrecrun; fitrec];

f1bestrecrun       = [f1bestrecrun;f1best]; %A record of main operating times

f2bestrecrun       = [f2bestrecrun;f2best]; %A record of grading margins

f3bestrecrun       = [f3bestrecrun;f3best]; %A record of backup operating times

mr

TMbestrecrun = [TMbestrecrun;TMbest];
```

```
   if bestgen < mr_bestever
      if sum(TMbest > 1.000000)==0 & sum(dt_best<-0.001) == 0 & sum(TMbest
<0.01)==0
         mr_bestever     = bestgen;
         mr_f1_bestever  = f1best;
         mr_f2_bestever  = f2best;
         mr_f3_bestever  = f3best;
         mr_TMbestever   = TMbest;
         fitrecbest      = fitrec;
      end
   end
end
fitrecrun_fin          = fitrecrun(:,maxgen);
[fitrecrun_sort,ind_fitrecbest]       = sort(fitrecrun_fin);
% fitrecbest            = fitrecrun(ind_fitrecbest(1),:);
fitrecrunave           = mean(fitrecrun,1);
fitrecrunstd           = std(fitrecrun(:,maxgen),1,1);
f1bestrecrunave        = mean(f1bestrecrun,1);
f1bestrecrunstd        = std(f1bestrecrun,1,1);
f2bestrecrunave        = mean(f2bestrecrun,1);
f2bestrecrunstd        = std(f2bestrecrun,1,1);
f3bestrecrunave        = mean(f3bestrecrun,1);
f3bestrecrunstd        = std(f3bestrecrun,1,1);
```

```
successrate          = (good_dt_lb_ub/mr)*100

%RESULTS

plot(fitrecrunave)

title('Average fitness')

figure

plot(fitrecbest)

title('Best fitness')
```