Nature Inspired Hybrid Optimization Algorithms for Load Flow Analysis of Autonomous Microgrids

by

Saad Mohammad Abdullah

MASTER OF SCIENCE

IN

ELECTRICAL AND ELECTRONIC ENGINEERING

Department of Electrical and Electronic Engineering
Islamic University of Technology (IUT)
Board Bazar, Gazipur-1704, Bangladesh
November, 2019

CERTIFICATE OF APPROVAL

The thesis titled 'Nature Inspired Hybrid Optimization Algorithms for Load Flow Analysis of Autonomous Microgrids' submitted by Saad Mohammad Abdullah, St. No. 161021012 of Academic year 2016-17 has been found as satisfactory and accepted as partial fulfillment of the requirement for the Degree MASTER OF SCIENCE IN ELECTRICAL AND ELECTRONIC ENGINEERING on November 08, 2019.

1.

Dr. Ashik Ahmed (Supervisor) — Chairman
Associate Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT), Gazipur.

2.

Dr. Md. Ruhul Amin — Ex-Officio
Professor and Head,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT), Gazipur.

3.

Dr. Golam Sarowar — Member
Associate Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT), Gazipur.

4. 08-11-2019

Dr. Md. Shahid Ullah — Member (External)
Professor and Head,
Department of Electrical and Electronic Engineering,
Daffodil International University (DIU), Dhaka.

Declaration of Candidate

It is hereby declared that this thesis report or any part of it has not been submitted elsewhere for the award of any Degree or Diploma and it has not been copied from other person's work.

Dr. Ashik Ahmed
Associate Professor,
Department of Electrical and Electronic Engineering,
Islamic University of Technology (IUT), Gazipur.
Date: November 08, 2019

Saad Mohammad Abdullah
Student No.:161021012
Academic Year: 2016-17
Date: November 08, 2019

Dedicated to my parents

for

their love and support

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DG	Distributed Generation
PCC	Point of Common Connection
VSI	Voltage Source Inverter
PLL	Phase Locked Loop
GA	Genetic Algorithm
DE	Differential Evolution
PSO	Particle Swarm Optimization
ICA	Imperialist Competitive Algorithm
FPA	Flower Pollination Algorithm
GOA	Grasshopper Optimization Algorithm
ICGA	Imperialist Competitive Genetic Algorithm
ICDE	Imperialist Competitive Differential Evolution
ICFPA	Imperialist Competitive Flower Pollination Algorithm
ICGOA	Imperialist Competitive Grasshopper Optimization Algorithm

ACKNOWLEDGMENTS

First and foremost, I would like to express my heartiest gratitude to the almighty Allah (SWT), for giving me the required capability and patience to conduct this research work.

I would like to express my sincere gratitude to my thesis supervisor, Dr. Ashik Ahmed for his continuous guidance and support throughout the course of this work. I am thankful to my supervisor for introducing me to the topics focused in this research. Whenever I encountered some problem, his guidance and suggestions helped me to have a better understanding on the related topics.

I would also like to thank all the faculty members of the department of EEE, IUT for their motivation and inspiration during this research work.

Finally, I am extremely grateful and express my deepest gratitude to my parents for their continuous moral support throughout this research work. Without their support and sacrifice it would never have been possible for me to make it this far. Special thanks to my elder sister, family members and friends for their motivation and support.

Saad Mohammad Abdullah

November, 2019

ABSTRACT

Load flow analysis is a significant tool for proper planning, operation and dynamic analysis of a conventional power system which provides the steady state values of voltage magnitudes and angles at fundamental frequency. However, due to the absence of slack bus in an autonomous microgrid, modified load flow algorithms should be adopted considering the system frequency as one of the solution variables. This work proposes the application of nature inspired hybrid optimization algorithms for solving the load flow problem of autonomous microgrids. Several nature-inspired algorithms such as, Genetic Algorithm (GA), Differential Evolution (DE) algorithm, Flower Pollination Algorithm (FPA) and Grasshopper Optimization Algorithm (GOA) are separately merged with Imperialistic Competitive Algorithm (ICA) to form four hybrid algorithms named as ICGA, ICDE, ICFPA and ICGOA and their performances are tested on a modified IEEE 37-Bus microgrid system as a case study. Particle swarm optimization (PSO) algorithm is also employed to perform the load flow analysis of the same case study system. Among the above-mentioned algorithms, to identify the algorithm with better performance, independent samples *t-tests* have been conducted using SPSS statistical analysis software. From the statistical analysis, it has been identified that ICDE exhibit better performance compared to the other algorithms in terms of the number of iterations and the execution time required to complete the optimization process for the load flow analysis.

Chapter 1

Introduction and Background

This chapter starts with a brief introduction of microgrid system, its different operating modes and the control scheme adopted for meeting load demand as presented in section 1.1. Section 1.2 contains a brief discussion regarding the importance of load flow analysis for proper planning and energy management of microgrid system. The literature review portion in section 1.3 contains information regarding the traditional methods of load flow analysis of a power system. The different methodologies and algorithms employed in various literatures for the load flow analysis of autonomous microgrid are also highlighted in this section. In light of the literature review the motivation towards this research is discussed in section 1.4 and the thesis objectives are outlined in section 1.5.

1.1 Introduction to Microgrid

Microgrid concept evolved from the idea of integrating distributed generation (DG) units along with energy storage elements and controllable loads [1]. As a result of growing energy demand and as a replacement to the aging infrastructure of current transmission and distribution system; the incorporation of DGs in the distribution level gained much popularity over the last few decades [2]. Renewable energy resources such as solar energy, wind energy, hydro power along with other generating sources like diesel engine, internal combustion engine, gas turbines, fuel cells etc. are some of the commonly used DG units. Storage devices such as batteries, energy capacitors, flywheels and different flexible loads are aggregated with the DG units to form a particular microgrid system [3]. The interconnection among these devices for a typical microgrid system is depicted in Fig. 1.1. In this figure; solar energy (photovoltaic modules), wind energy (wind turbines) and generating sources (diesel engine) are shown as the DG units and a battery is used to represent the energy storage element. With the help of the renewable energy resources electricity can be produced at low fuel cost with less carbon emission. Energy storage elements also play a vital role in achieving energy balance in case of load fluctuations [4]. In terms of the nature of the energy produced, each DG unit has to be associated with power electronic interfaces (DC/AC or AC/DC/AC) in order to connect the DG unit to a particular bus in the

electrical network [5]. The final stage of these interfaces consists of dc/ac inverters. Thus, the control process associated with the inverter has significant importance on the operation of a microgrid system [6]. To design different controllers, it is often necessary to determine the steady state operating points of a system. Load flow analysis is a significant tool in determining the steady state operating points. Load flow analysis also provide useful information for proper monitoring, operation and energy management of a power system. Thus, over the years, several studies have been conducted to propose efficient methods of load flow analysis of microgrid systems in order to ensure proper planning, operation and control.

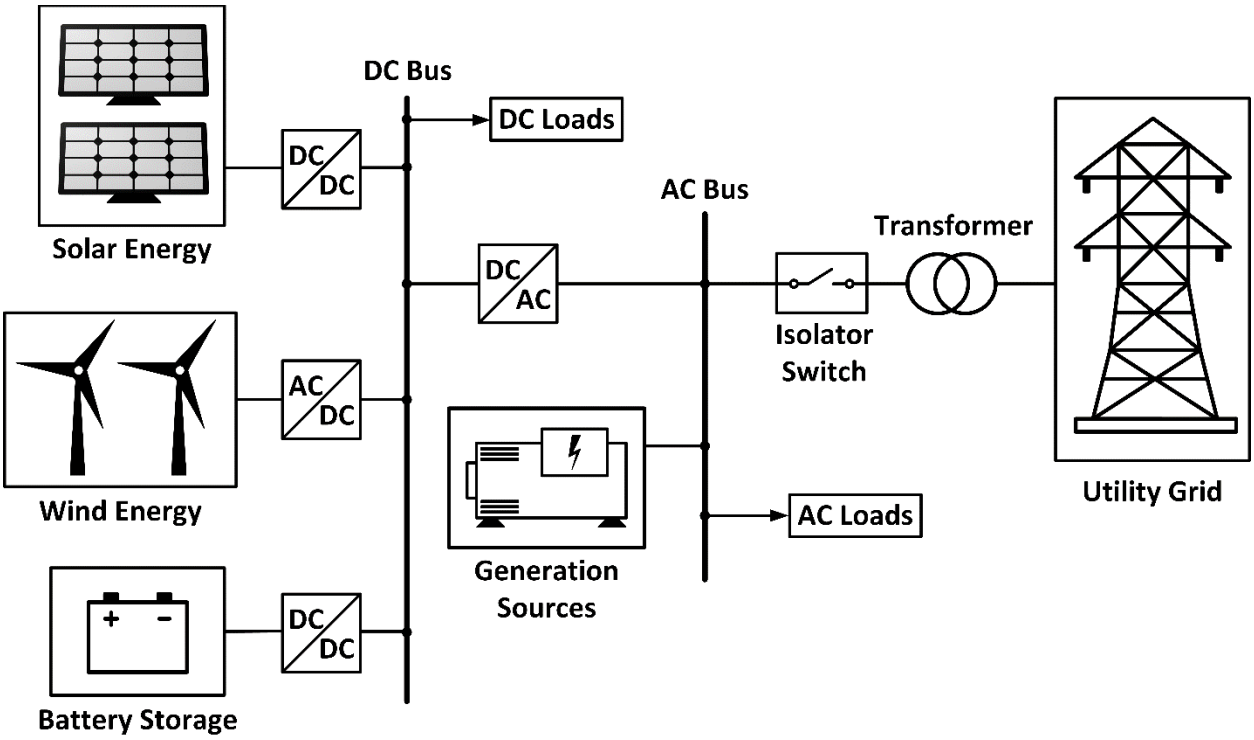


Fig. 1.1: Architecture of a typical microgrid system

1.1.1 Operating Modes of Microgrid System

Based on the position of the isolator switch a microgrid system can have two operation modes: grid-connected (online) mode and autonomous (islanded) mode [7].

Grid-connected Mode: In grid connected mode, the microgrid system is coupled with the utility grid. The coupling of the microgrid with the main grid can be achieved by closing the isolator switch shown in Fig. 1.1. The isolator switch also signifies the point of common connection

(PCC) between the DGs and the utility grid. In grid connected mode, the power supplied by the DG units are almost independent of the load fluctuations because the spinning reserve associated with the online generators compensate the unbalance between the generated power and the electrical power consumption. Thus, the voltage and frequency at the PCC are maintained by the main grid. Furthermore, as the capacities associated with the DGs connected in a microgrid is small compared to the utility grid, the disturbances in the microgrid very rarely has significant effect on the frequency adjustment of the grid [8].

Autonomous (Islanded) Mode: In autonomous mode, the microgrid system operates as an independent entity with isolation from the utility grid. The operation of microgrid can be switched to autonomous or islanded mode by keeping the isolator switch open as shown in Fig. 1.1. In order to achieve a stable operating condition, all the DGs have to take the responsibility of maintaining the balance between power supply and load variations. Due to the low capacities of the DG units and as they are power electronically interfaced, the spinning reserve concept cannot be employed for stable operation in autonomous mode. Dedicated control scheme is required for islanded microgrid to provide voltage and frequency control establishing real and reactive power balance [9]. According to the change in load demand the voltage and frequency set points to the DG units has to be adjusted on regular basis. Thus, establishing a control process to ensure reliability of operation of microgrid in case of autonomous mode becomes a challenging task. Droop control schemes are most commonly employed on the power electronic inverters associated with each DG unit to obtain the regulation of voltage and frequency [10, 11].

1.1.2 Droop Control Method for Autonomous Microgrid

As mentioned in earlier section, the DG units in the microgrid system are interfaced with power electronic interfaces and the final stage of each interface consist of an dc/ac inverter such as voltage source inverter (VSI). Each VSI is associated with inner current controllers and outer voltage controllers. To control the VSIs, real power-frequency $(P-\omega)$ and reactive power-voltage $(Q-V)$ droop control methods are used to imitate the behavior of synchronous generator [12-15]. The voltage and frequency of the VSI has to be regulated in such a way that power demand for all critical loads within the microgrid are met adequately. If there is increase in load demand, then the DG units should generate more power by slightly reducing the frequency of the VSI as per the $(P-\omega)$ droop control scheme. Similarly, to control the flow of

reactive power with respect to the load demand the voltage magnitude is adjusted following the $(Q-V)$ droop control scheme. Thus, in case of load fluctuations, the frequency and voltage magnitude are adjusted according to the required power demand in order to ensure reliability of operation. The droop co-efficients are set in such a way that all the DG units in a microgrid system can share the total load demand with respect to their individual power ratings [15, 16]. The decentralized droop control method is a very effective primary control strategy for islanded microgrids. Unlike centralized microgrid control system, the droop control scheme does not require inter-unit communication between the DGs. Droop controller associated with each DG operate independently based on locally measured values in order to obtain appropriate power sharing [17, 18].

1.2 Importance of Load Flow Analysis

Load flow analysis is an integral part of power system analysis. It is also a prerequisite for transient stability analysis, optimal power flow and contingency studies [19]. The bus voltage magnitudes and phase angles along with the active and reactive power flowing through the transmission lines are the key information obtained from load flow analysis. This information is important for proper monitoring of the present status of a network. It is also important for necessary planning prior to setting-up a new system and to ensure optimal operation and future expansion of existing system [20]. Solutions obtained from load flow analysis can be used to find steady-state operating points of a particular system. Then a system model with a set of nonlinear equations can be linearized around the steady-state operating points [21]. Due to the small capacity of the DG units in case of autonomous microgrid, a single DG unit cannot act as the infinite bus, rather all the DGs have to regulate their voltage and frequency to meet the required load demand. Thus, load flow analysis plays a significant role in assessing the feasibility of autonomous operation under specified system constraints. Proper energy management and power sharing among the DGs and the overall stability analysis of microgrid system can also be facilitated by efficient load flow methodology [22].

1.3 Literature Review

Over the years; Gauss-Seidel (GS), Newton-Raphson (NR) methods have been widely used for efficient and reliable load flow analysis of power systems [23, 24]. Several studies showed that Newton-Raphson method possess better convergence characteristics, but with a higher

computational time. In order to achieve faster computation, the decoupled and fast decoupled load flow techniques were proposed as modified versions of the traditional NR method [25, 26]. According to different researches it has been shown that, it becomes difficult to achieve convergence using NR method or fast decoupled (FD) method in case of ill-conditioned distribution networks such as radial network, networks with low X/R ratios or in case of unbalanced distributed loads [27, 28]. As a result, several modified versions of the GS, NR, FD load flow methods have been proposed in different literatures. Another popular method is the backward/forward sweep (BFS) method which performs the load flow analysis using forward and backward sweeps through the network based on basic electrical circuit laws such as Kirchoff's voltage and current laws [29, 30]. Later on, different modified versions were proposed by introducing quadrating equations to calculate voltage magnitudes as proposed in some literatures. In other studies, power summation and admittance summation methods were introduced in BFS load flow analysis. However, the different backward/forward sweep methods of load flow analysis were designed to solve radial distribution networks. For weakly meshed distribution networks, compensation based load flow methods were proposed in different literatures [31-33]. The main concept of compensation based algorithm is to introduce several mesh break points in order to represent the weakly meshed network as a single radial network. The load flow analysis of this equivalent radial network can then be performed following the process of backward/forward sweep method [34].

As an alternative to the conventional load flow methodologies, evolutionary computation was introduced in the load flow analysis of power systems. The evolutionary algorithms are derivative-free in nature as there is no requirement to calculate the Jacobian matrix. Furthermore, these algorithms are independent of the initial settings of the solution variables and have the capability to generate multiple solutions. In [35], a constrained genetic algorithm (CGA) was proposed for load flow analysis of power systems by minimizing the active and reactive power mismatches in case of PQ buses and minimizing the mismatch between active power and voltage requirements in case of PV buses. Later on, advanced constrained genetic algorithm (ACGA) was proposed by Wong et al. in order to improve the performance of CGA [36]. Two acceleration techniques were introduced in ACGA to facilitate faster convergence. In first step, the current population was updated by nodal voltage differential technique and then a percentage of the updated population is further accelerated using gradient technique in the second step. Ting

et al. developed a hybrid CGA/PSO algorithm as a modification of ACGA [37]. In this hybrid algorithm, PSO was introduced to replace the differential voltage technique used in ACGA to update the current population. A multi-objective differential evolution (MODE) algorithm was introduced in [38], in order to optimize the balance between real and reactive power. In order to find the most optimum solution from the pareto optimal solutions, a fuzzy membership and pseudo-weight vector approach was introduced in that technique.

The abovementioned algorithms were developed to perform load flow analysis of conventional power systems or distribution networks. These load flow methods assume the system frequency to be constant implying the concept of slack bus. In case of autonomous operation of microgrid system, the concept of slack bus in not applicable as all the DG units need to collectively regulate the voltage and frequency as per the load demand. Rather than considering frequency to be constant, it has to be calculated as one of the load flow variables. However, conventional concepts of power flow analysis were used in case of an autonomous microgrid by treating the local bus of the generating unit with the highest power rating as the slack bus [39, 40]. Some studies conducted by Kamh et al. were based on the application of single phase backward-forward sweep algorithm for single phase networks and sequence-components frame power flow solver for three phase networks [41, 42]. The accuracy of these methods is limited due to the approximation of constant frequency throughout the solution. Furthermore, the decentralized droop control-based operation of microgrid system was not considered in these studies. In order to compensate the shortcomings of the conventional methods, several approaches have been proposed considering the frequency as one of the power flow solution variables. In the work of Abdelaziz et al. [22], a Newton-trust region method was proposed to perform the load flow analysis of three phase systems considering that some of the DGs are governed by the droop control method. Later on, a modified Gauss Seidel (MGS) method and a modified Newton-Raphson (MNR) methods were proposed in the work of Mumtaz et al. to perform the load flow analysis of islanded microgrids focusing on the droop characteristics of DGs [43, 44]. However, for these methods, the microgrid system model was developed in stationary reference frame considering the voltages and currents as phasors which only allowed steady state analysis of the system and failed to provide necessary information for obtaining the linearized dynamic model of the system. In the study conducted by Mueller and Kimball [21], the system model was developed in synchronous reference frame and a quasi-Newton method was introduced to solve

the load flow analysis considering the system frequency, reference frame angles and voltage magnitudes as the load flow variables. Most of these load flow techniques use gradient-based algorithms which require evaluation of derivatives for a series of complex equations. Gradient based techniques often fail to obtain a global solution as these algorithms mostly converge on a local solution depending upon the selection of the initial starting point [45].

Multi-solution based evolutionary algorithms have better possibility of avoiding a local optimum by exploring a larger portion of the search space [46]. For droop-controlled islanded microgrid, a load flow algorithm was introduced where particle swarm optimization (PSO) technique was used to determine the droop parameters [47]. Later on, Abedini [48] applied hybridized ICGA algorithm for load flow analysis by incorporating imperialist competitive algorithm (ICA) with the multi-solution based genetic algorithm (GA). Fairly good performance was obtained in the aforementioned work; however, the system modeling was done in stationary reference frame.

1.4 Motivation towards the Research

The literature review gives an insight regarding the necessity of developing modified load flow methodologies for autonomous microgrids. Several approaches have been proposed by different researchers for this purpose as highlighted in the literature review. Some of these methods were based on the assumption of constant system frequency which is contradictory to the characteristics of autonomous microgrids. In some techniques the droop control scheme of microgrids was not considered. Most of these approaches were focused on gradient-based approaches which have a tendency of getting stuck in a local optimum if the initial starting point is not selected close to the global optimum. Furthermore, these derivative based approaches often fail to converge in case of nonlinear and discontinuous functions. As a result, nature inspired metaheuristic optimization algorithms were introduced for the load flow analysis of autonomous microgrids. However, only a few researches have been conducted in this regard such as the application of ICGA algorithm as discussed in the literature review. Thus, it can be considered that there is still room for further exploration regarding the application of nature inspired optimization algorithms in case of autonomous microgrids. Considering this fact in this research, a comparative study will be demonstrated among ICGA and three other hybrid algorithms ICDE, ICFPA and ICGOA where the differential evolution (DE) algorithm, flower pollination algorithm (FPA) and grasshopper optimization algorithm (GOA) will be separately merged with

ICA. Furthermore, for this study, synchronous reference frame-based system model is adopted for the droop controlled autonomous microgrid. Developing the system model in synchronous reference frame provides multiple advantages such as transforming the time variant quantities into time invariant ones which makes the modeling of different controllers easier. In this research, the modified IEEE-37 bus system is considered as a case study system. For a comparative study along with the aforementioned four hybrid algorithms load flow analysis of the modified IEEE-37 bus system will be also performed using particle swarm optimization (PSO). From the comparative study the algorithm with the better performance will be identified which can be considered as a prospective stochastic technique for non-conventional load flow methodology for autonomous microgrids.

1.5 Thesis Objectives

The main goal of this thesis is to develop a load flow methodology for autonomous microgrids based on the nature-inspired optimization algorithms. In order to meet that goal the following objectives have been considered for this thesis

- Study of different nature-inspired optimization algorithms
- To propose several hybrid optimization algorithms based on the study conducted
- Application of these hybrid algorithms in performing load flow analysis of autonomous microgrids
- To perform comparative study on the results obtained through different hybrid algorithms
- To identify the algorithm with better performance among the proposed ones

1.6 Outline of this thesis

Chapter 2 demonstrates the mathematical model of a microgrid system in the synchronous reference frame. As each DG unit is coupled with an inverter, the mathematical model of an inverter along with its necessary controllers are presented in this chapter. The development of load and line equations and bus voltage equations are also shown in this chapter.

In Chapter 3, discussions are carried out on some of the nature-inspired optimization algorithms considered for this research which are the genetic algorithm (GA), particle swarm optimization (PSO), differential evolution (DE), imperialist competitive algorithm (ICA), flower pollination

algorithm (FPA) and grasshopper optimization algorithm (GOA). This chapter gives a brief insight to the optimization process of these algorithms along with respective pseudocodes.

In Chapter 4, the formulations necessary for the load flow analysis is presented. The process of forming four hybrid algorithms are also discussed in this chapter. The steps involved in the optimization process of GA, DE, FPA and GOA are separately merged with the optimization process of ICA to form the four hybrid algorithms.

The application of these hybrid algorithms in performing the load flow analysis is demonstrated in Chapter 5. The modified IEEE-37 bus system was considered as the case study system for this study. The comparison among the results obtained through different hybrid algorithms are shown in this chapter.

Lastly, Chapter 6 contains some concluding remarks and a brief discussion on the future research scope.

Chapter 2

Mathematical Model of Microgrid

Multiple distributed generation (DG) units are aggregated in a microgrid system. In most of the cases direct connection of these DGs to the distribution network is not suitable due to the nature of energy produced. Thus, before connecting to a bus, the DGs are associated with power electronic interfaces such as inverters [49]. As a result, developing mathematical model of the inverter along with its associated controllers is important for the analysis of microgrid systems. The control strategy of an inverter coupled with an individual DG is shown diagrammatically in Fig. 2.1. The discussion in this section describes the dynamic model of droop-controlled inverters along with the necessary load and line equations to develop the complete microgrid model. The modeling technique described in this section is based on the studies carried out in [21, 50, 51], where the system model was developed in the synchronous reference frame instead of the stationary reference frame following the Park's transformation technique. As a result of the transformation from stationary reference frame to synchronous reference frame, three phase quantities can be converted to two phase which reduces the complexity of the system. In addition to that time varying quantities can be converted to time invariant ones as a result of this transformation which makes the design of different controllers easier as regular PI controllers can be used.

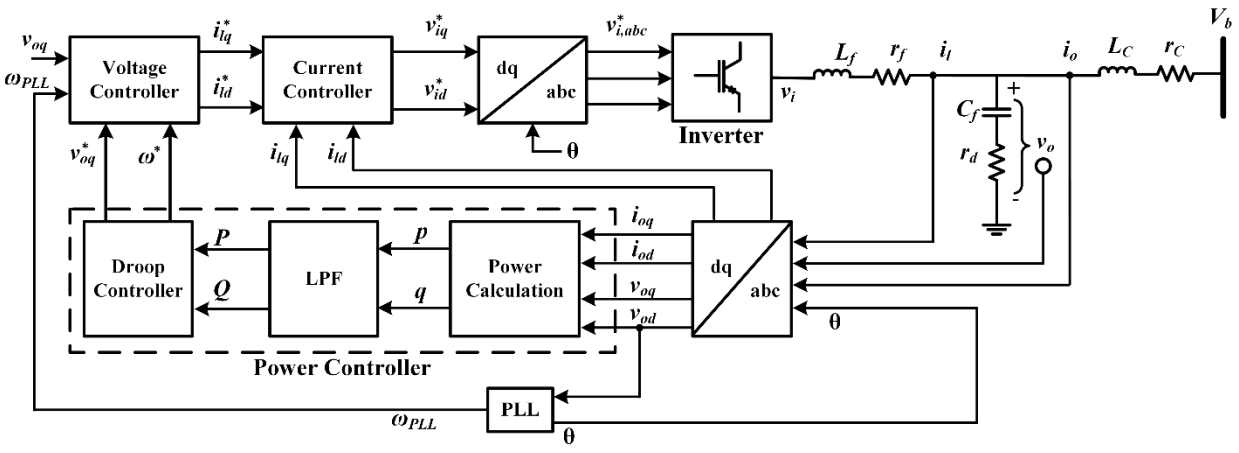


Fig. 2.1: Block diagram of control strategy of droop-controlled inverter for individual DG

2.1 Inverter Model

2.1.1 Reference Frame Transformation

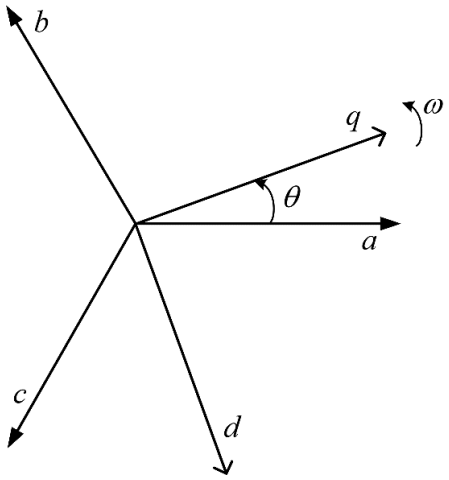


Fig. 2.2: Relationship between *abc* and *dq* quantities

The inverter is coupled to the inverter bus through an LCL filter as shown in Fig. 2.1. The three-phase capacitor voltage and inductor currents of the LCL filter are transformed from stationary *abc* reference frame to the synchronously rotating *dq* reference frame following the theory of Park's transformation. The axes of the three-phase stationary *abc* reference frame and the direct (*d*) and quadrature (*q*) axes of the synchronously rotating *dq* reference frame are shown in Fig. 2.2, where θ represents the angle difference between the two reference frames and ω represents the rotational speed of the *dq* reference frame. This transformation is accomplished using the following equation

$$\begin{bmatrix} v_{oq} \\ v_{od} \\ v_{o0} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} cos(\theta) & cos(\theta - \frac{2\pi}{3}) & cos(\theta + \frac{2\pi}{3}) \\ sin(\theta) & sin(\theta - \frac{2\pi}{3}) & sin(\theta + \frac{2\pi}{3}) \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} v_{oa} \\ v_{ob} \\ v_{oc} \end{bmatrix} \tag{2.1}$$

where, v_{oq} and v_{od} are respectively the *q-axis* and *d-axis* components of the filter capacitor voltage and v_{oa}, v_{ob} and v_{oc} are voltages in the stationary reference frame. The reference frame transformation of the filter inductor current, i_l and the output current, i_0 can be obtained using

similar relationships. In equation (2.1), the transformation angle *(θ)* is calculated by a phase locked loop (PLL). Details of this transformation technique is given in [52].

2.1.2 Phase Locked Loop (PLL)

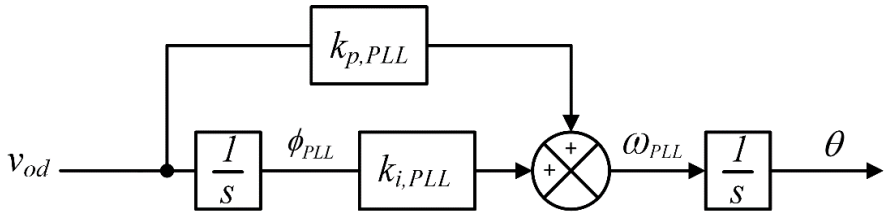


Fig. 2.3: *dq*-based PLL

A *dq*-based PLL is used to measure the phase and frequency. A proportional-integral (PI) controller is associated with the PLL as shown in Fig. 2.3. The PLL is used to force the *d-axis* component of the capacitor voltage to become 0. This results in the steady-state voltage magnitude to be equal to its *q-axis* component. The PLL equations are

$$\frac{d}{dt}\phi_{PLL} = -v_{od} \tag{2.2}$$

$$\omega_{PLL} = -k_{p,_{PLL}} v_{od} + k_{i,_{PLL}} \phi_{PLL} \tag{2.3}$$

$$\frac{d}{dt}\theta = \omega_{PLL} \tag{2.4}$$

where, ϕ_{PLL} is the integrator state of the PI controller. $k_{p,_{PLL}}$ and $k_{i,_{PLL}}$ are respectively the proportional and integral gain, ω_{PLL} is the calculated frequency and θ is the transformation angle.

2.1.3 Power Controller

First of all, the instantaneous active (*p*) and reactive (*q*) powers are calculated in the power controller based on the values of the capacitor voltage and output current. The instantaneous active (*p*) and reactive (*q*) power outputs are given by

$$p = \frac{3}{2}(v_{od} i_{od} + v_{oq} i_{oq}) \tag{2.5}$$

$$q = \frac{3}{2}(v_{oq} i_{od} - v_{od} i_{oq}) \tag{2.6}$$

Then, average active (P) and reactive (Q) power values are calculated by passing the instantaneous power outputs through a first order low pass filter (LPF). The filter equations are

$$\frac{d}{dt} P = \omega_c p - \omega_c P \tag{2.7}$$

$$\frac{d}{dt} Q = \omega_c q - \omega_c Q \tag{2.8}$$

where, ω_c is the cut-off frequency of the low pass filter. The process of active and reactive power calculations is shown in Fig. 2.4.

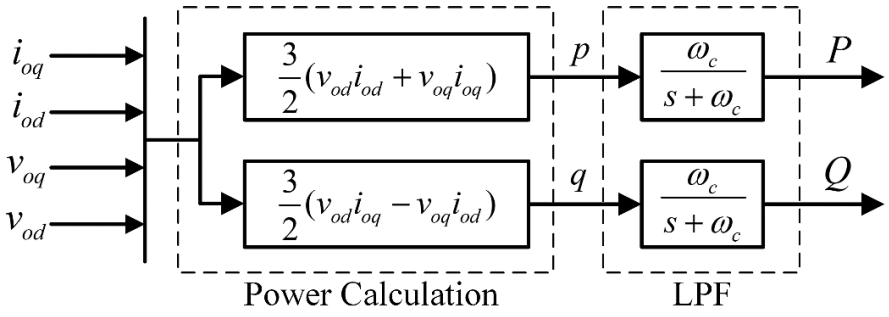


Fig. 2.4: Calculation of active and reactive power

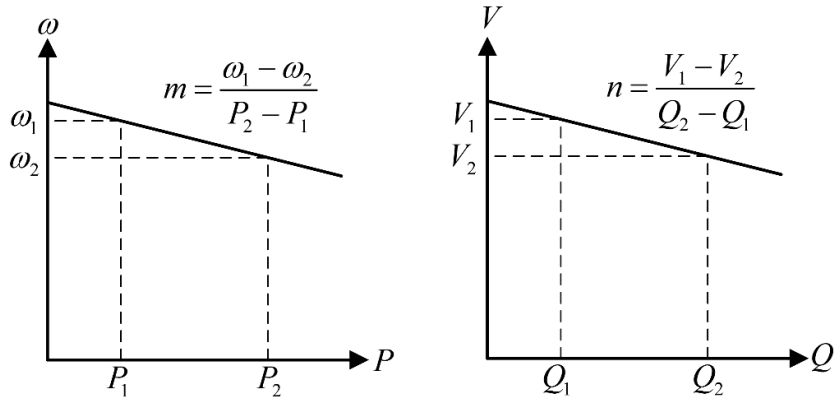


Fig. 2.5: Droop characteristics curves

Then, the droop controller generates the voltage magnitude and frequency references depending upon the calculated values of the active and reactive powers. The *P*-ω and *Q*-*V* droop equations are used to generate the frequency reference, ω^* and *q-axis* voltage magnitude reference, v_{oq}^* respectively. The characteristics of the droop curves are shown in Fig. 2.5.

$$\omega^* = \omega_n - mP \tag{2.9}$$

$$v_{oq}^* = V_n - nQ \tag{2.10}$$

where, ω_n represents the nominal frequency set point and V_n represents the nominal set point of the *q-axis* output voltage. The droop constants m and n are calculated from specified range of frequency and voltage magnitude.

$$m = \frac{\omega_{max} - \omega_{min}}{P_{max}} \tag{2.11}$$

$$n = \frac{V_{oq,max} - V_{oq,min}}{Q_{max}} \tag{2.12}$$

2.1.4 Voltage Controllers

The voltage controller compares between the reference and measured values of frequency and voltage, and generates the reference values of the output filter inductor currents through a pair of PI controller as shown in Fig. 2.6. The voltage controller equations are

$$\frac{d}{dt}\varphi_d = \omega^* - \omega_{PLL} \tag{2.13}$$

$$i_{ld}^* = k_{pv,d}(\omega^* - \omega_{PLL}) + k_{iv,d}\varphi_d \tag{2.14}$$

$$\frac{d}{dt}\varphi_q = v_{oq}^* - v_{oq} \tag{2.15}$$

$$i_{lq}^* = k_{pv,q}(v_{oq}^* - v_{oq}) + k_{iv,q}\varphi_q \tag{2.16}$$

where, φ_d and φ_q represent the integrator states of the voltage controllers. The proportional and integral gains of the respective *d-axis* and *q-axis* controllers are represented by $k_{pv,d}$, $k_{pv,q}$, $k_{iv,d}$ and $k_{iv,q}$.

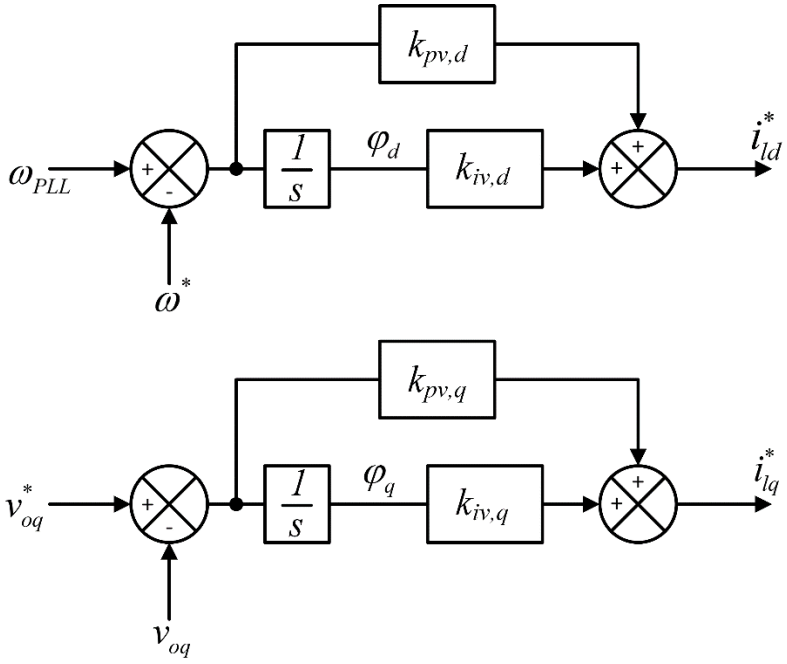


Fig. 2.6: Voltage controllers

2.1.5 Current Controllers

The reference values of filter inductor current are compared with the measured filter inductor current using the current controllers. As outputs reference values of voltages are provided by these current controllers, which are used to generate switching signals for the inverter. As shown in Fig. 2.7, two PI controllers are used for this purpose. The cross-coupling terms appearing due to the reference frame transformation are also eliminated by these controllers. The current controller equations are

$$\frac{d}{dt}\gamma_d = i_{ld}^* - i_{ld} \tag{2.17}$$

$$v_{id}^* = k_{pc,d}(i_{ld}^* - i_{ld}) + k_{ic,d}\gamma_d - \omega_n L_f i_{lq} \tag{2.18}$$

$$\frac{d}{dt}\gamma_q = i_{lq}^* - i_{lq} \tag{2.19}$$

$$v_{iq}^{*} = k_{pc,q}(i_{lq}^{*} - i_{lq}) + k_{ic,q}\gamma_q + \omega_n L_f i_{ld} \tag{2.20}$$

where, γ_d and γ_q are the integrator state of the current controllers. $k_{pc,d}$, $k_{pc,q}$, $k_{ic,d}$ and $k_{ic,q}$ represent the proportional and integral gains of the *d-axis* and *q-axis* controllers respectively.

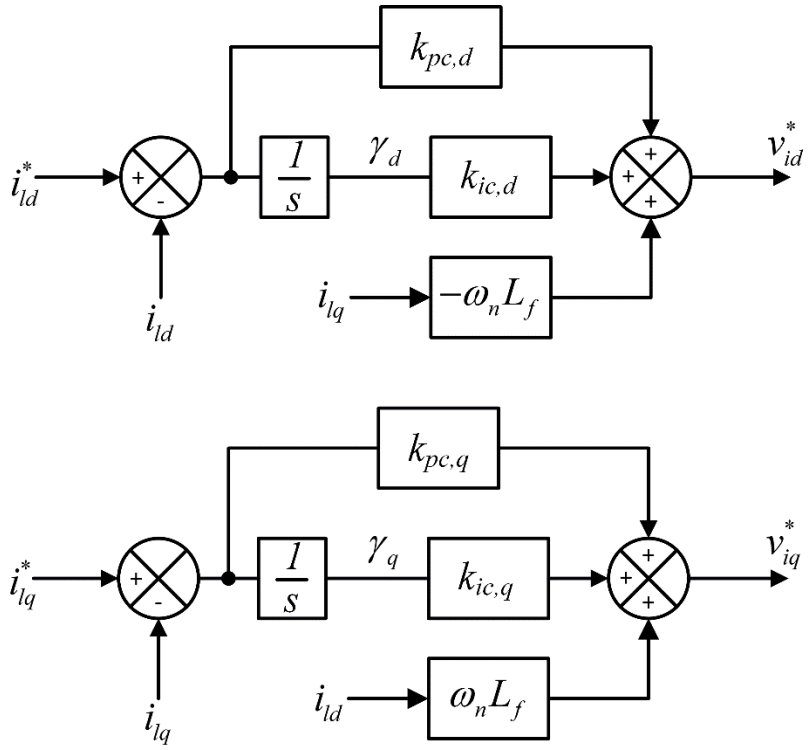


Fig. 2.7: Current controllers

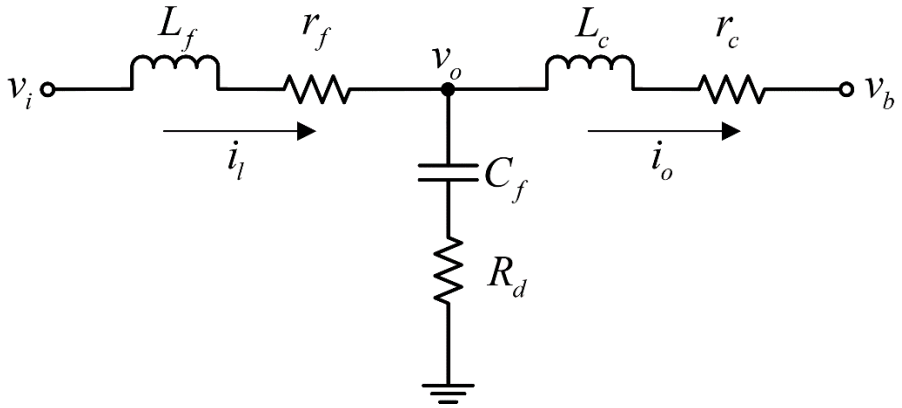


Fig. 2.8: Output LCL filter

2.1.6 LCL Filter Equations

The inverter output is connected to the microgrid through an LC filter and coupling inductor as demonstrated in Fig. 2.8. The filter inductor L_f, filter capacitor C_f and coupling inductor L_c collectively form the LCL filter. The parasitic resistance of these components is also considered for the inverter model as shown in Fig. 2.1. The filter dynamics are governed by the following equations

$$\frac{d}{dt}i_{ld} = \frac{1}{L_f}(-r_f i_{ld} + v_{id} - v_{od}) + \omega^* i_{lq} \tag{2.21}$$

$$\frac{d}{dt}i_{lq} = \frac{1}{L_f}(-r_f i_{lq} + v_{iq} - v_{oq}) - \omega^* i_{ld} \tag{2.22}$$

$$\frac{d}{dt}i_{od} = \frac{1}{L_c}(-r_c i_{od} + v_{od} - v_{bd}) + \omega^* i_{oq} \tag{2.23}$$

$$\frac{d}{dt}i_{oq} = \frac{1}{L_c}(-r_c i_{oq} + v_{oq} - v_{bq}) - \omega^* i_{od} \tag{2.24}$$

$$\frac{d}{dt}v_{od} = \frac{1}{C_f}(i_{ld} - i_{od}) + \omega^* v_{oq} + R_d \frac{d}{dt}(i_{ld} - i_{od}) \tag{2.25}$$

$$\frac{d}{dt}v_{oq} = \frac{1}{C_f}(i_{lq} - i_{oq}) - \omega^* v_{od} + R_d \frac{d}{dt}(i_{lq} - i_{oq}) \tag{2.26}$$

In equations (2.23) and (2.24), v_{bd} and v_{bq} represent the bus voltages at the grid side of the coupling inductor.

2.1.7 Local to Global Reference Frame Transformation

Each inverter model is developed in its own local reference frame. For modeling a microgrid system with several inverters, it is necessary to translate the values defined in the local reference frame of an inverter to a common reference frame called the global reference frame. This concept can be visualized from Fig. 2.9, where *DQ* reference frame is considered to be the common reference frame and *dq* reference frames indicate the local reference frames of the

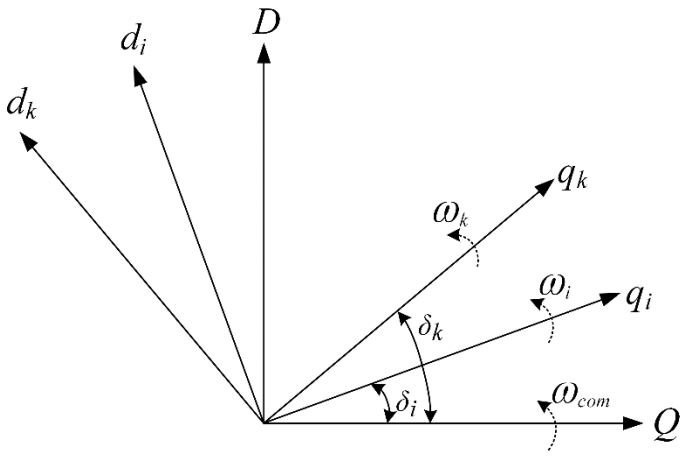


Fig. 2.9: Relationship between global reference frame and local reference frames

inverters in the system. In Fig. 2.9, *i* indicates the number of inverters connected in the system where, *i = 1, 2, … …, k*. This transformation can be achieved by

$$\begin{bmatrix} F_q \\ F_d \end{bmatrix} = R(\delta) \begin{bmatrix} f_q \\ f_d \end{bmatrix} \tag{2.27}$$

$$R(\delta) = \begin{bmatrix} cos\delta & -sin\delta \\ sin\delta & cos\delta \end{bmatrix} \tag{2.28}$$

where, δ is the angular difference between local and global reference frame. In equation (2.27), lowercase letter is used to indicate local reference frame and uppercase letter is used to indicate global reference frame. The angle δ is defined by

$$\frac{d}{dt}\delta = \omega - \omega_{PLL} \tag{2.29}$$

where, ω is the frequency of the global reference frame and ω_{PLL} is the frequency measured by PLL of a particular inverter. Often, the reference frame of the first inverter in the system is chosen as the global reference frame. In this work, we set $\omega_1 = \omega_{PLL}$ and $\delta_1 = 0$, which implies $\dot{\delta}_1 = 0$. For other inverters, $\omega \neq \omega_{PLL}$ and δ has to be calculated following equation (2.29).

2.2 Equations for Load and Line

To complete modeling the entire microgrid model, it is necessary to formulate the state equations for load and line in the global reference frame. Loads can be of constant impedance type which is basically a combination of resistors and inductors (*RL* loads) as depicted in Fig. 2.10. The equations of *RL* load connected to the i^{th} bus can be described by

$$\frac{d}{dt} I_{load,d_i} = \frac{1}{L_{load_i}} (V_{bd_i} - R_{load_i} I_{load,d_i}) + \omega I_{load,q_i} \tag{2.30}$$

$$\frac{d}{dt} I_{load,q_i} = \frac{1}{L_{load_i}} (V_{bq_i} - R_{load_i} I_{load,q_i}) - \omega I_{load,d_i} \tag{2.31}$$

Line currents between two adjacent buses *i* and *j* connected through a transmission line can be described by

$$\frac{d}{dt} I_{line,d_{ij}} = \frac{1}{L_{line_{ij}}} (V_{bd_i} - V_{bd_j} - R_{line_{ij}} I_{line,d_{ij}}) + \omega I_{line,q_{ij}} \tag{2.32}$$

$$\frac{d}{dt} I_{line,q_{ij}} = \frac{1}{L_{line_{ij}}} (V_{bq_i} - V_{bq_j} - R_{line_{ij}} I_{line,q_{ij}}) - \omega I_{line,d_{ij}} \tag{2.33}$$

where, $0 \le i < j \le N$. N represents the total number of buses in the system.

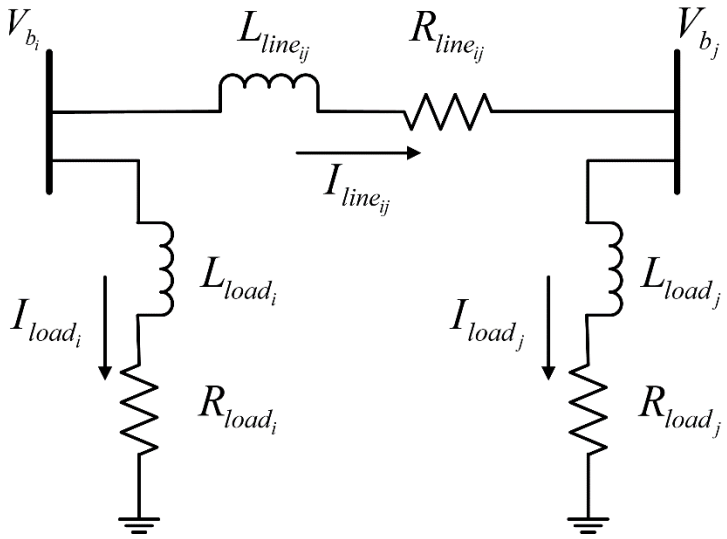


Fig. 2.10: Line configuration between two buses

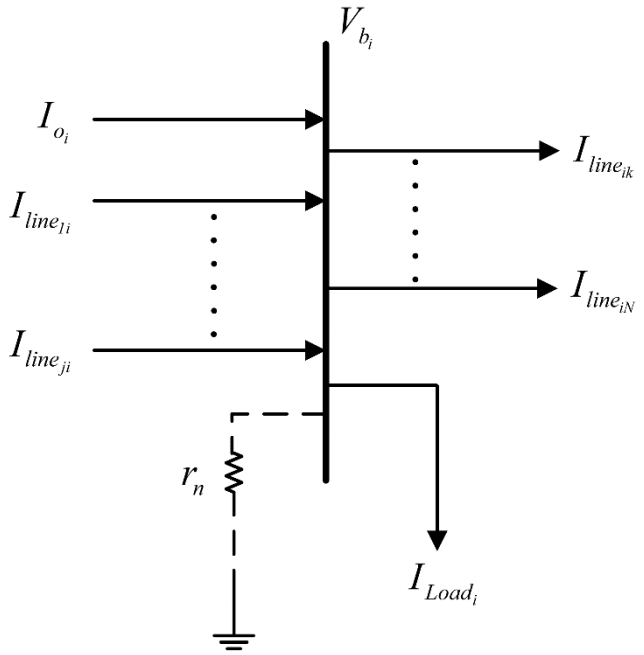


Fig. 2.11: Line and load currents at a particular bus

2.3 Bus Voltage Equations

The virtual resistance method can be used to find the expression of the bus voltages in the global reference frame. To determine the bus voltage at the i^{th} bus using virtual resistance method, a high resistance connection is considered between bus *i* and ground. This high resistance actually represents open circuit. Ideally this resistance should be infinite, but typically a large value of resistance is considered for modeling purpose.

Bus voltage expressions are typically dependent on any incoming inverter output current, incoming or outgoing line currents and currents flowing through the connected load as shown in Fig. 2.11. The voltage expression at bus *i* can be expressed as

$$V_{bd_i} = r_n (I_{od_i} - I_{load,d_i} + \sum_{j=1}^{i-1} I_{line,d_{ji}} - \sum_{k=i+1}^{N} I_{line,d_{ik}}) \tag{2.34}$$

$$V_{bq_i} = r_n (I_{oq_i} - I_{load,q_i} + \sum_{j=1}^{i-1} I_{line,q_{ji}} - \sum_{k=i+1}^{N} I_{line,q_{ik}}) \tag{2.35}$$

where, $j \le i < k \le N$.

2.4 Overall Microgrid Model

The equations of the inverter, load and line model described so far can be used to represent the overall microgrid model. The state vector of a droop-controlled inverter connected to the i^{th} bus can be formulated as

$$x_{inv_i} = \begin{bmatrix} \delta_i & P_i & Q_i & \varphi_{d_i} & \varphi_{q_i} & \gamma_{d_i} & \gamma_{q_i} & i_{ld_i} & i_{lq_i} & v_{d_i} & v_{q_i} & i_{od_i} & i_{oq_i} & \phi_{PLL_i} \end{bmatrix}^T \tag{2.36}$$

If there are total p number of inverters connected to the system, then the combined state vector of all the inverters can be represented as

$$x_{inv} = \begin{bmatrix} x_{inv_1} & x_{inv_2} & \dots & \dots & x_{inv_p} \end{bmatrix} \tag{2.37}$$

Considering the equations of the load model described from equations (30) and (31), the state vector of a load connected at the i^{th} bus is

$$x_{load_i} = \begin{bmatrix} I_{load,d_i} & I_{load,q_i} \end{bmatrix}^T \tag{2.38}$$

If there are total N number of buses in the microgid system and one specific RL load is connected to each bus, then the combined state vector of all the loads will be

$$x_{load} = \begin{bmatrix} I_{load,d_1} & I_{load,q_1} & \dots & \dots & I_{load,d_N} & I_{load,q_N} \end{bmatrix}^T \tag{2.39}$$

The equations of line current from equations (2.32) and (2.33) can be used to represent the states of a line between buses i and j as

$$x_{line_{ij}} = \begin{bmatrix} I_{line,d_{ij}} & I_{line,q_{ij}} \end{bmatrix}^T \tag{2.40}$$

The overall state vector of all the lines can be represented as

$$x_{line} = \begin{bmatrix} I_{line,d_{1j}} & I_{line,q_{1j}} & \dots & \dots & I_{line,d_{kN}} & I_{line,q_{kN}} \end{bmatrix}^T \tag{2.41}$$

where, $1 < j \leq k \leq N$.

Based on the combined state vectors of the droop-controlled inverters, loads and lines; the states of the overall microgrid model can be described as

$$x_{mg} = \begin{bmatrix} x_{inv} & x_{load} & x_{line} \end{bmatrix} \tag{2.42}$$

State-space model of the whole microgrid system will have the following form

$$\dot{x}_{mg} = A_{mg} x_{mg} + B_{mg} u_{mg} \tag{2.43}$$

The elements of the state matrix, A_{mg} and input matrix, B_{mg} is defined by equations (2.2), (2.7), (2.8), (2.13), (2.15), (2.17), (2.19), (2.21) to (2.26), (2.29) and (2.30) to (2.33). The input vector, u_{mg} can be represented in terms of the bus voltages as

$$u_{mg} = \begin{bmatrix} V_{bd_1} & V_{bq_1} & \dots & \dots & V_{bd_N} & V_{bq_N} \end{bmatrix}^T \tag{2.44}$$

Chapter 3

Introduction to Nature-inspired Optimization Algorithms

Optimization has become an important tool for solving various problems in the field of engineering, science, technology, industrial designs and even for business studies. Most of the real-world problems can be formulated in terms of nonlinear equations with some constraints associated. These constraints or restrictions can be associated in the form of resources, cost, time or any other parameters upon which the objective function is dependent. Thus, robust optimization techniques are required to find the most suitable solution to a particular problem without violating the constraints. In general, the optimization techniques are often classified as either deterministic optimization techniques or stochastic optimization techniques.

Deterministic optimization techniques are mostly calculus-based or derivative-based approaches. These are single-solution based approaches and the search is directed towards an optimum point based on the gradient of the function. These optimization techniques are highly dependent on the selection of the initial starting point or the initial guess. If the initial guess is not selected in the neighborhood of the global optimum, then the solution is likely to entrap to a local optimum. Apart from that, application of these algorithms is dependent upon the existence of derivatives. In case of nonlinear objective functions having discontinuities, the gradient-based algorithms are not supposed to work [53].

On the other hand, stochastic optimization methods operate based on some random search techniques. Different evolutionary and swarm intelligence-based algorithms are categorized under the stochastic methods. These algorithms are also referred to as nature-inspired optimization algorithms because the search techniques used in these algorithms are motivated from some natural phenomenon. Most of these techniques are multi-solution based optimization process where a set of random solutions are initially generated. Then, through a series of random operators the solution sets are updated at each iteration with the focus to move towards the global optimum. Thus, through random trial and error process these algorithms try to reach the optimum solution. Hence, all the stochastic algorithms are also termed as metaheuristic algorithms.

Exploitation and exploration are two major components of these nature-inspired optimization algorithms. Exploration is the process of generating diverse solutions along the search space which is necessary to identify other prospective solutions apart from the current solutions. This process is referred to as a global searching process. On the other hand, exploitation refers to local searching mechanism which is focused on searching around the neighborhood of the current solutions. A balance between these two operators is necessary to obtain an overall global optimum solution. As these algorithms are free from the calculation of derivatives and also not completely dependent upon the initial starting point, the metaheuristic algorithms gained much popularity over the gradient-based techniques [45, 54].

The following sections contain discussions regarding the optimization process of some of the nature-inspired optimization algorithms opted for this research. The algorithms are namely; the genetic algorithm (GA), particle swarm optimization (PSO), differential evolution (DE), imperialist competitive algorithm (ICA), flower pollination algorithm (FPA) and grasshopper optimization algorithm (GOA). Formation of hybrid algorithms by merging these individual algorithms is presented in the subsequent chapter.

3.1 Genetic Algorithm (GA)

Genetic algorithm (GA) is one of the most popular evolutionary optimization algorithms which was inspired by Charles Darwin's theory of natural evolution. The concept of GA was first proposed in [55], based on the theory of natural selection and survival of the fittest which refers to the selection of the fittest individuals to participate in the reproduction of the next generation. The initial versions of genetic algorithms were developed by introducing some form of binary coding to represent the solution variables. Thus, for multidimensional genetic algorithms each parameter (variable) of a solution set is represented as binary bit-string. It is worth mentioning that in analogy to natural evolution, the solution sets are considered as set of chromosomes / population individuals in case of GA and each parameter is considered as genes characterizing the chromosomes / individuals. The initial solution sets are termed as the parent chromosomes. For generating offspring, two parent chromosomes are randomly selected and their corresponding bit patterns are exchanged at random points referring to the crossover process of natural reproduction. The generated offspring undergo mutation process to add diversity to the population individuals. After crossover and mutation, the total population size increases and

based on the fitness value of each individual, a specific number of individuals are selected as parents to generate offspring in the next iteration [56]. Although, the binary coded genetic algorithms are very robust search techniques for global optimization, but a major drawback is the higher computational cost specially in case of multidimensional problems as each parameter has to be encoded with a corresponding bit-sequence. To address this issue, real-coded genetic algorithm was first proposed in [57], where each possible solution sets were considered as the chromosomes and each real valued parameter were considered as the genes. The crossover and mutation operators were also designed in terms of real values of the variables. Over the years, in order to improve the performance of real coded genetic algorithms, researchers have put much emphasis on the development of sophisticated real coded crossover operators as can be found from the works described in [58, 59]. The pseudo code for GA is shown in Algorithm 1 and the steps involved in the optimization process is described as follows.

Step 1: Random generation of the initial population *(X)* of *n* number of possible solutions (parent chromosomes) within the lower (*lb)* and upper *(ub)* boundaries of the search space.

Step 2: Initialization of the percentage of crossover (p_c) and the percentage of mutation (p_m). Thus, determining the number of offspring (n_c) and the number of mutants (n_m) to be generated from the parents in each iteration.

Step 3: Initialization of mutation rate (μ_m) and the extension rate for crossover (γ).

Step 4: Generation of offspring (child chromosomes) by performing the crossover operation. Extended-line crossover is considered for this case as shown in Fig. 3.1. First of all, two parent chromosomes (P_1, P_2) are randomly selected from the initial population (*X*). A random number (λ) is generated based on the value of γ and two child chromosomes (y_1, y_2) are generated following the equations presented in Algorithm 1. The range of possible offspring is shown by an extended straight line connecting the two parent chromosomes as indicated in Fig. 3.1 for two-dimensional space. As two child chromosomes are generated from each crossover operation, the inner loop of crossover is executed $n_c/2$ number of times to generate total n_c number of offspring. After each execution of the crossover operation, the information of the child chromosomes (y_1, y_2) are stored in a variable, *Y*.

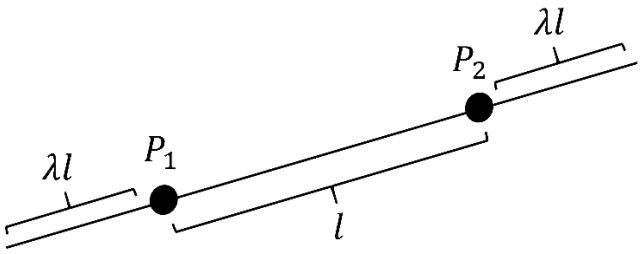


Fig. 3.1: Extended-line crossover with range of possible offspring

Step 5: Performing Gaussian mutation to generate the mutants. First, a random parent chromosome (P) is selected. Then, the dimension(s), j at which the mutation is going to be performed is randomly selected with respect to the mutation rate (μ_m) and the parameter(s) at the j^{th} *dimension(s) of P* will be perturbed through the mutation process as shown in Algorithm 1. Where, $gaussian(0, \frac{ub-lb}{10})$ is a function to generate random Gaussian number with zero mean and the standard deviation of $\left(\frac{ub-lb}{10}\right)$. The inner loop of mutation is executed for n_m number of times and at each execution the generated mutant (z) is stored in a variable, *Z*.

Step 6: The total population size is updated by merging the generated offspring (Y) and mutants (Z) with the initially generated parent chromosomes (X). For each candidate solution in the total population the objective function is evaluated to determine the fitness of each solution. The possible solutions are sorted based on their fitness value and fittest n number of solutions are selected as parent chromosomes (X) for the next generation.

Step 7: If the stopping criterion is satisfied, then the solution with the best fitness is considered as the optimum solution. Otherwise, the process will continue from step 4 until the termination criterion is satisfied.

Algorithm 1 Pseudo code for GA

Begin:
Generate the initial population (chromosomes): $X_i \sim U(\text{lb}, \text{ub})$ ($i = 1, 2, \ldots, n$)
Initialize the percentage of crossover (p_c) and mutation (p_m)
Determine the total number of offspring (n_c) and total number of mutants (n_m)
Initialize mutation rate, μ_m
Initialize the extension rate for crossover, γ

while(the stopping criterion is not satisfied)
for *i* = 1 to $n_c/2$
Randomly select two parent chromosomes P_1, P_2
$Y \leftarrow$ Initialize to store the offspring
Crossover (P_1, P_2, γ)
{
$\lambda \leftarrow$ a random number in [$-\gamma$, $1+\gamma$, *size*(P_1)]
$y_1 = \lambda P_1 + (1 - \lambda) P_2$
$y_2 = \lambda P_2 + (1 - \lambda) P_1$
}
Update Y
end
$Z \leftarrow$ Initialize to store the mutants
for *i* = 1 to n_m
Randomly select a parent chromosome *P*
Mutation (P, μ_m)
{
$z = P;$
$j \leftarrow$ randomly select the number of dimensions to be mutated with respect to μ_m
$z(j) = P(j) + gaussian(0, \frac{ub-lb}{10})$
}
Update Z
end
Total population, *T = [X; Y; Z]*
Evaluate fitness of each solution set in total population $f(T_i)$ ($i = 1, 2, \ldots, n + n_c + n_m$)
Update *X* with the fittest n number of solution sets
Best Solution = X(1)
end
return *Best Solution*
End

3.2 Differential Evolution (DE)

Differential Evolution (DE) is a real-parameter optimization algorithm which also falls into the category of evolutionary algorithms. It was first introduced in the work of Storn and Price [60]. Like other evolutionary algorithms DE solves a particular optimization problem in an iterative process by improving the candidate solutions in each iteration with respect to the objective function. As used in other evolutionary algorithms such as genetic algorithm (GA); genetic operators such as crossover, mutation, selection is also performed in case of DE. However, these operators are employed in a different manner. For example, in case of DE, the mutation operation is performed to perturb all the components of a solution vector. Whereas, in case of

GA, only a selected number of components in the candidate solution is perturbed. In DE, the optimization process starts with a randomly generated initial population. The solution vectors present in the population at the beginning of a particular generation are referred as parent vectors or genomes. With the help of mutation and crossover operation offspring are generated from the parent vectors. Each parent vector in the population first undergo a differential mutation process and the vector obtained from the mutation process is termed as the mutant vector. For a particular parent vector in the population; the differential mutation is performed by randomly selecting three other distinct solution vectors. Then, a scaled difference is taken between any two of these three vectors and the scaled difference is added with the third vector to obtain the mutant vector. Finally, offspring is generated from the mutant and parent vector by exchanging components of the parent and mutant vector on the basis of crossover probability. The offspring vector generated after crossover is known as the trial vector. The fitness of all the trial vectors are compared with the fitness of the parent vectors and the fittest solution vectors are chosen as parents for reproduction in the next generation keeping the population size constant [61]. Algorithm 2 contains the pseudo-code of DE and the step by step optimization process of DE is briefly discussed as follows.

Step 1: Initialization of the population *(X)* with *n* number of random solution vectors (parent chromosomes) within the lower (*lb)* and upper *(ub)* boundaries of the search space.

Step 2: Evaluation of the fitness value of each parent vector, $f(\mathrm{X}_i)$. Where, *i = 1, 2, …, n*.

Step 3: Initialization of the crossover probability (p_{CR}) in the range $[0, 1]$.

Step 4: Generation of parent vectors for the next generation.

> **Step 4.1:** Generation of the mutant vector (V_i) for the i^{th} parent vector by performing the differential mutation operation. From the *n* number of solution vectors of the current population three solution vectors are randomly chosen in order to create the mutant vector. Let us consider, X_j, X_k and X_l are the three randomly selected solution vectors where, $j, k, l \in [1, n]$ and $i \neq j \neq k \neq l$. Now, from these three vectors, difference between any two of them is scaled by a scaling factor, *F* and added with the third one to generate the mutant vector as indicated in the formula shown in Algorithm 2. Typically, the value of *F* is randomly drawn in the range between 0.4 and 1.

Step 4.2: In this step, the crossover operation is performed to recombine the mutant with the parent vector and generate offspring known as the trial vector (U_i). The exchange of components between the parent and mutant vector is dependent upon the crossover probability (p_{CR}). In binomial crossover, a random integer (m_{rand}) is generated in the range $[1, D]$, where D is the dimension of each solution vector. The m^{th} $(m \in [1, D])$ component of the trial vector, $U_i(m)$ will be taken from the m^{th} component of mutant vector, $V_i(m)$ when the value of a m_{rand} equals the value of m or when a randomly generated number, $rand[0, 1]$ is less than or equal to the crossover probability (p_{CR}). Otherwise, the m^{th} $(m \in [1, D])$ component of the trial vector, $U_i(m)$ will be set to the value of the m^{th} component of the parent vector, $X_i(m)$.

Step 4.3: Selection among the parent and the trial vector will be carried out in this step. If the fitness value of the trial vector (U_i) is better compared to the fitness of the parent vector (X_i), then the parent vector will be replaced by the trial vector. Otherwise, the parent vector will sustain for the next generation.

Step 4.4: Repeat steps 4.1, 4.2 and 4.3 for all the *n* number of parent vectors in the current generation.

Step 5: Determine the solution with the best fitness value among all the solution vectors in the current iteration.

Step 6: If the stopping criterion is satisfied, then the solution with the best fitness in the current iteration is considered as the optimum solution. Otherwise, the optimization process will continue from step 4.

Algorithm 2 Pseudo code for DE

Begin:
Initialize the population of solution vectors (genomes): $X_i \sim U(\text{lb}, \text{ub})$ (*i = 1, 2, …, n*)
Evaluate fitness of each solution vector: $f(\text{X}_i)$ (*i = 1, 2, …, n*)
Initialize the crossover probability $p_{CR} \in [0, 1]$

while(the stopping criterion is not satisfied)
for $i = 1\ to\ n$
Parent vector $\rightarrow X_i$
$j, k, l \leftarrow$ randomly select three integers in the range $[0, n]$ (where, $i \neq j \neq k \neq l$)
$F \leftarrow$ Scaling factor (randomly chosen in the range $[0.4, 1]$)

Generate Mutant vector: $V_i = X_j + F(X_k - X_l)$
$m_{rand} \leftarrow$ randomly chosen integer in the range $[1, D]$ ($D \rightarrow$ dimension of each solution vector)
$U_i \leftarrow$ initialize Trial vector with all components equal to zero
for $m = 1\ to\ D$
if $m == m_{rand}\ or\ rand[0,1] \leq p_{CR}$
$U_i(m) = V_i(m)$
else
$U_i(m) = X_i(m)$
end
end
Calculate fitness of Trial vector: $f(U_i)$
if $f(U_i) < f(\mathrm{X}_i)$
$\mathrm{X}_i = U_i$
else
X_i will remain unchanged
end
end
$Best\ solution =$ solution vector with best fitness in current iteration
end
return $Best\ solution$
End

3.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is a swarm intelligence-based optimization algorithm which was first developed in 1995 by Kennedy and Eberhart [62]. This algorithm was mainly inspired by the swarming behavior of creatures such as birds, fishes or bees which live in large colonies. The movement and intelligence of these swarms and their social interaction are the basis particle swarm optimization. In PSO, the members of the swarm are commonly referred as particles which move around a search space in order to find the best solution. The algorithm starts by random initialization of the positions of these particles. Then, through continuous iteration, the algorithm searches for optimum solution by updating the position of each particle based on the particle velocity, the personal best solution ($pbest$) and the global best solution ($gbest$). For a continuous optimization process, $pbest$ refers to the best solution a particle has achieved up to current iteration and $gbest$ is the overall best solution among all the particles in the current population. In each iteration, in terms of predefined inertia weight (w) and acceleration constants (C_1, C_2), the resultant velocity of each particle is calculated depending upon its previous velocity and the locations of $pbest$ and $gbest$. Thus, the resultant velocity guides the particles to move

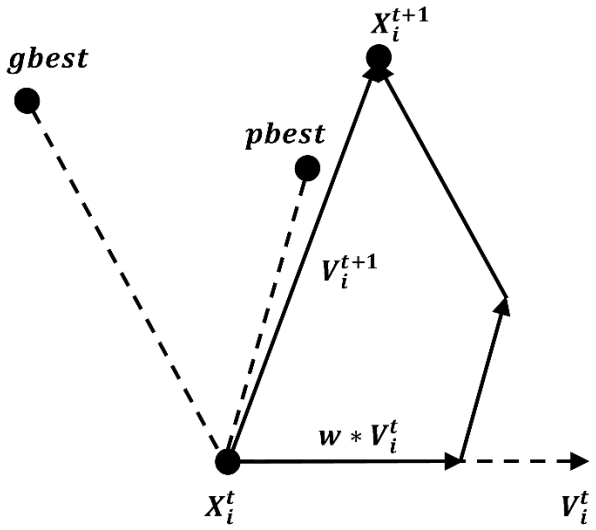


Fig. 3.2: Vector diagram relating PSO equations

towards the locations of its $pbest$ and $gbest$ [63]. In modified version of PSO a constriction factor (*K*) based approach was used to update the velocity in each iteration [64, 65]. In this approach, the constriction factor (*K*) is associated with the inertia weight and acceleration constants in updating the velocity. The pseudocode of PSO is presented in Algorithm 3 and the step by step procedure of applying PSO in solving an optimization problem is briefly discussed as follows.

Step 1: Randomly generating the initial positions of the particles *(X)* within the lower (*lb)* and upper *(ub)* boundaries of the solution variables.

Step 2: Evaluation of the fitness values of the particles $f(X)$. For the initial population, the $pbest$ solution of each particle is set to be same as the initial positions. Whereas, the $gbest$ solution is the solution with the best fitness among the initial generation.

Step 3: Initialization of acceleration constants, C_1 and C_2. Calculating the constriction factor (*K*) following the equation shown in Algorithm 3. Initially, the inertia weight (w) is set equal to *K*.

Step 4: Randomly generating the initial velocity vectors (V) of the particles.

Step 5: Calculating the velocity of the particles following the equation shown in Algorithm 3 in order to update the positions of the particles for the next iteration. In this equation, the acceleration constants, C_1 and C_2 influence the movement of the particles in the search space before reaching the target region. Along with these two constants the concept of inertia weight

(w) was introduced to obtain a balance between exploration and exploitation [66]. Furthermore, in order to enhance the probability of convergence the concept of constriction factor (K) was introduced.

Step 6: Updating the position of each particle by adding the velocity vector with the position vector of each particle. This process can be visualized from the vector diagram presented in Fig. 3.2.

Step 7: Evaluation of the fitness of the updated particles. Based on the calculated fitness values the location of $pbest$ and $gbest$ are updated. The inertia weight (w) is also updated based on the inertia weight damping factor (w_{damp}).

Step 8: Checking the stopping condition. If the stopping condition is satisfied, then the location of $gbest$ is considered as the optimum solution. Otherwise, the optimization procedure is repeated from step 5.

Algorithm 3 Pseudo code for PSO

Begin:
Randomly initialize the population (particles): $X_i \sim U(\text{lb}, \text{ub})$ ($i = 1, 2, \ldots, n$);
Evaluate fitness of each particle: $f(X_i)$ ($i = 1, 2, \ldots, n$);
Initialize best solution for each particle for initial population: $pbest_i = X_i$;
Determine, $gbest =$ Best solution among all the particles in current population;
Initialize inertia weight damping factor (w_{damp}) and acceleration constants C_1 and C_2;
Calculation of constriction factor: $K = \dfrac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}$ (where, $\varphi = C_1 + C_2$ & $\varphi > 4$);
Initialize inertia weight, $w = K$;
Randomly initialize the velocity of each particle: V_i

while(the stopping criterion is not satisfied)
for i = 1 to n
  Calculate velocity: $V_i^{t+1} = w * V_i^t + K * C_1 * rand(0,1) * (pbest_i^t - X_i^t) + K * C_2 * rand(0,1) * (gbest^t - X_i^t)$;
  Update position of particle: $X_i^{t+1} = \mathrm{X}_i^t + V_i^{t+1}$;
  Evaluate fitness of the updated particle: $f(X_i^{t+1})$;
  if $f(X_i^{t+1}) < f(pbest_i^t)$
    $pbest_i^{t+1} = X_i^{t+1}$;
    if $f(X_i^{t+1}) < f(gbest^t)$
      $gbest^{t+1} = X_i^{t+1}$;
    end

```
    end
    w = w * w_damp;
end
end
return gbest
End
```

3.4 Imperialist Competitive Algorithm (ICA)

Imperialism is the policy of a country to establish its dominance over other countries through political, economic or military power. The more developed countries try to exercise their power over underdeveloped countries and possess the control over their resources. Thus, an empire is formed where the developed country act as the imperialist and the countries under its control are termed as the colonies to the imperialist. Going back in history, it can be seen that several empires existed throughout the centuries. There always existed a competition among different empires to take control over the colonies of other empires to enhance its own power with the ultimate target of ruling the whole world. Apart from that, the developing colonies also try to liberate them from the authority of the imperialist to regain control over their own resources. In this course, a particular developing colony may become a threat for the imperialist and eventually by taking complete authority this colony may emerge as the new imperialist of the empire. Thus, the whole phenomenon is a game of survival of the fittest where the most powerful countries dominate over the others. Inspired from this concept of imperialistic competition, the imperialist competitive algorithm (ICA) was first introduced by Atashpaz-Gargari and Lucas [67]. This algorithm is referred as social counterpart of genetic algorithm (GA) as it is based on human social evolution compared to the biological evolution in case of GA. Similar to other evolutionary algorithms, ICA starts with a randomly generated initial population and the population individuals are termed as countries in this case. Based on the fitness values of the countries, a specific number of countries are set as the imperialists and the rest of the countries are allotted as colonies to the imperialists. Then, the positions of the colonies are moved towards their respective imperialist following a process called assimilation. The assimilation process imitates the fact that the imperialists try to force their cultural beliefs and customs to the colonies in order to have a better control over them. However, some colonies resist to follow the forcibly pressed customs of the imperialists. This scenario is mimicked through a process called revolution, where the positions of some of the colonies are randomly perturbed. If a colony is

found to possess better fitness than the imperialist, then the roles of imperialist and colony are interchanged. Then, the total power of each empire is computed and imperialistic competition is performed by picking the weakest colony from the weakest empire and assigning them to the empire which has the most likelihood to possess that colony [68]. This process continues until a convergence criterion is not satisfied. The pseudo code of ICA is given in Algorithm 4 and the optimization process is discussed through the following steps.

Step 1: Defining input parameters; the population size (n_{pop}), number of imperialists (n_{imp}) and number of colonies (n_{col}).

Step 2: Initialization of the population by randomly generating the positions of n_{pop} number of countries *(X)* within the lower (*lb)* and upper *(ub)* boundaries of the solution variables.

Step 3: Computing the cost or fitness of each country and sorting them in terms of the fitness value. Selecting the fittest n_{imp} number of countries as imperialists (Imp). The remaining n_{col} number of countries will be assigned as colonies to these imperialists.

Step 4: The maximum cost (C_{max}) among the selected imperialists is identified and the cost (C) of each imperialist is normalized with respect to this maximum cost. From the computed values of normalized costs (C_n), the normalized power (P_n) of each imperialist is calculated following the equation as indicated in Algorithm 4. From the total number of colonies (n_{col}), the initial number of colonies (NC) to be assigned to a particular imperialist is calculated from its respective normalized power.

Step 5: Generation of each empire by randomly selecting NC number of colonies from the total number of colonies and assigning them as colonies (Col) to their respective imperialist (Imp).

Step 6: Initialization of assimilation coefficient (β), probability of revolution (p_R), revolution rate (μ).

Step 7: In this step, the positions of the colonies are updated by moving the colonies towards their respective imperialists. This process in referred to as Assimilation. How much a particular colony will move towards the imperialist depends upon the assimilation co-efficient (β) and the distance (d) between the colony and imperialist. Each colony is moved towards the imperialist by x units where x is a uniformly distributed random number in the range $[0, \beta * d]$.

Step 8: In this step, some of the colonies are randomly selected and the positions of these colonies are updated by randomly selecting the dimension(s) at which the parameter is going to be perturbed. In case of ICA, this process of perturbation at random dimensions of the colonies is termed as Revolution. Whether a particular colony will undergo revolution or not is determined with respect to the probability of revolution (p_R) and the dimensions (l) at which the perturbation is going to be performed is randomly selected based on the revolution rate (μ). As indicated in Algorithm 4, the l^{th} parameter of the m^{th} colony of the j^{th} imperialist is going to be perturbed by a random number having gaussian distribution with zero mean and standard deviation of $(\frac{ub-lb}{10})$.

Step 8: Intra-empire competition is performed in this step, where the cost of each colony is compared with the cost of the imperialist. If any colony has a better cost compared to the imperialist, then the position of the imperialist and colony are exchanged.

Step 9: Calculating the total cost (TC) of an empire by adding the scaled value of the mean fitness of the colonies with the fitness of the imperialist.

Step 10: Imperialistic competition is performed to select the weakest colony from the weakest empire and assign it to the empire which has the most likelihood to possess it. This depends on the possession probability (P_p) of each empire which is calculated from the total cost (TC). First the total cost of each empire is normalized, then from the normalized values the possession probability is calculated as shown in the pseudo code in Algorithm 4.

Step 11: The imperialist with the best fitness value is termed as the $Best\ solution$ in the current iteration.

Step 12: If the stopping criterion is satisfied, then the $Best\ solution$ in the current iteration is considered as the optimum solution. Otherwise, the optimization process will continue from step 7.

Algorithm 4 Pseudo code for ICA

Begin:
Initialize n_{pop} (population size), n_{imp} (number of imperialists) and n_{col} (number of colonies);
Generate the initial population (countries): X_i $(i = 1, 2, \dots, n_{pop})$;
Evaluate fitness or cost of each country: $C_i = f(X_i)$ $(i = 1, 2, \dots, n_{pop})$;

Sort the countries with respect to their costs and select best n_{imp} number of countries among them as imperialists (Imp);
Determine the maximum cost among the imperialists: $C_{max} = \max_i\{C_i\} \quad (i = 1, 2, \dots, n_{imp})$;
Normalize the cost of each imperialist: $C_{nj} = C_{max} - C_j \quad (j = 1, 2, \dots, n_{imp})$;
Determine the normalized power of each imperialist: $P_{nj} = \left| \frac{C_{nj}}{\sum_i^{n_{imp}} C_i} \right| \quad (j = 1, 2, \dots, n_{imp})$;

Compute initial number of colonies to be assigned to each imperialist:

$$NC_j = round(P_{nj} * n_{col}) \quad (j = 1, 2, \dots, n_{imp});$$

Randomly select NC_j number of colonies (Col_j) and assign them to the j^{th} imperialist;
Initialize β (assimilation coefficient), p_R (probability of revolution), μ (revolution rate),
ξ (mean cost co-efficient of the colonies);
while(the stopping criterion is not satisfied)

Assimilation:
for j = 1 to n_{imp}
  for m = 1 to NC_j
    $d \leftarrow$ distance between colony and imperialist;
    $x \sim U(0, \beta * d)$;
    $Col_{jm} = Col_{jm} + x$;
  end
end

Revolution:
for j = 1 to n_{imp}
  for m = 1 to NC_j
    $l \leftarrow$ randomly select the number of dimensions to be perturbed with respect to μ;
    if $rand[0, 1] \leq p_R$
      $Col_{jm}(l) = Col_{jm}(l) + gaussian(0, \frac{ub-lb}{10})$;
    end
  end
end

Intra-empire competition:
Evaluate the fitness of updated colonies;
for j = 1 to n_{imp}
  for m = 1 to NC_j
    if $f(Col_{jm}) < f(Imp_j)$
      Exchange the position of Imp_j and Col_{jm};
    end
  end
end

Total power of an empire:
for j = 1 to n_{imp}
$TC_j = f(Imp_j) + \xi * mean\{f(Col_j)\}$;
end

Imperialistic competition:
Maximum total cost among the empires: $TC_{max} = \max_i\{TC_i\} \quad (i = 1, 2, \dots, n_{imp})$;
Calculate normalized total cost of each empire: $TC_{nj} = TC_{max} - TC_j \quad (j = 1, 2, \dots, n_{imp})$;
Obtain possession probability of each empire: $P_{p_j} = \left| \frac{TC_{nj}}{\sum_i^{n_{imp}} TC_i} \right| \quad (j = 1, 2, \dots, n_{imp})$;
Pick the weakest colony from the weakest empire and allocate it to the empire which has the most likelihood to possess it based on possession probability (P_p);

$Best\ solution =$ imperialist with the best fitness in current iteration
end
return $Best\ solution$
End

3.5 Flower Pollination Algorithm (FPA)

Inspired by the pollination process of flowers, Flower Pollination Algorithm (FPA) was first proposed in [69]. Pollination process in flowers can primarily be classified as Biotic and Abiotic pollination. Biotic pollination process requires the help of pollinators in the form of insects, birds or bats to transfer pollen from one flower to another. Whereas, in abiotic pollination, pollen is transferred through the help of wind or diffusion in water without the necessity of pollinators. About 90% of pollination process in flowering plants occur in biotic form and only 10% pollination takes place in abiotic process. Flower constancy is also an important term in flower pollination. It refers to the fact that many pollinators travel to flowering plants of certain species in search of nectar. This ensures the transfer of pollen to the same or conspecific species and increases the probability of reproduction of the same flower species. Pollination process in flowers can be further classified as self-pollination and cross-pollination. Self-pollination process takes place through the transfer of pollen to the same flower or another flower in the same plant. On the other hand, cross-pollination refers to the process of transferring pollen from one plant to another plant. Thus, biotic and cross-pollination can be regarded as global pollination as the insects or pollinators traverse a long distance. Whereas, abiotic and self-pollination is referred as local pollination as the pollination takes place within a small neighborhood [70]. In FPA, a

global searching process is introduced mimicking the concept of global pollination and flower constancy where Levy distribution is normally used to indicate the jump or fly distance of pollinators in the global pollination process [71]. This global searching mechanism enhances diversity among the solution sets within the population. Moreover, the concept of local pollination and flower constancy is utilized in developing a local search operator. This local search operator also plays an important role of exploiting the search area in the vicinity of current solution. The pseudo-code of FPA is shown in Algorithm 5 and the steps involved in the optimization process is described as follows.

Step 1: Random initialization of the initial population *(X)* of *n* number of candidate solutions (flowers) within the lower (*lb)* and upper *(ub)* boundaries of the solution vector. For simplicity, normally it is considered that each flower produces only one pollen gamete. Thus, each solution is considered to be a flower or a pollen gamete.

Step 2: Evaluation of the fitness value of each flower in the population and identification of the best solution (g^*) within the initial population.

Step 3: Initialization of switch probability (p) by choosing a value between 0 to 1. Whether a solution set will undergo local pollination or global pollination depends upon this value of p.

Step 4: Generation of candidate solutions for the next iteration.

> **Step 4.1:** If a generated random number is greater than the value of p, then global pollination is performed according to the formula of global pollination written in Algorithm 5. In global pollination, the insects have to make large jumps or have to move a long distance. To address this behavior of insects, Levy distribution is adopted to generate a step vector, L.
>
> **Step 4.2:** If the random number generated in the previous step is less than p, then local pollination is performed following the formula of local pollination shown in Algorithm 5. From the total number of solutions in the population; j^{th} and k^{th} solution sets are randomly selected for local pollination. The value of ϵ is randomly selected from a uniform distribution in the range [0, 1].
>
> **Step 4.3:** After performing either local or global pollination, if the newly generated solution (X_i^{t+1}) has better fitness compared to the previous one (X_i^t), then the previous

solution is replaced with the new one. Otherwise, the previous solution (X_i^t) will remain unchanged.

Step 4.4: Repeat steps 4.1, 4.2 and 4.3 to go through the pollination process for all the n number of solutions in the population in the current generation.

Step 7: The solution with the best fitness among the current solutions is identified and g^* is updated accordingly.

Step 8: The termination criterion is checked in this step. If the criterion is satisfied then the current best solution, g^* is returned as the optimum solution. Otherwise, the optimization process will be continued from step 4.

Algorithm 5 Pseudo code for FPA

Begin:
Generate the initial population (flowers): $X_i \sim U(\text{lb}, \text{ub})$ (i = *1, 2, …, n*)
Evaluate fitness of each flower: $f(X_i)$ (i = *1, 2, …, n*)
Determine, $g^* =$ Best solution among the initial population
Initialize the switch probability between local and global pollination $p \in [0, 1]$

while(the stopping criterion is not satisfied)
for i = 1 to n
  if $rand[0, 1] > p$
    Generate a step vector, L following Levy distribution (dimension will be same as a particular solution)
    Perform global pollination: $X_i^{t+1} = X_i^t + L(g^* - X_i^t)$
  else
    $\epsilon \leftarrow$ randomly select a number from a uniform distribution in $[0, 1]$
    $j, k \leftarrow$ randomly select two numbers from the total number of solutions
    Perform local pollination: $X_i^{t+1} = X_i^t + \epsilon(X_j^t - X_k^t)$
  end
Calculate fitness of new solution: $f(X_i^{t+1})$
  if $f(X_i^{t+1}) < f(X_i)$
    $X_i = X_i^{t+1}$
  else
    X_i will remain unchanged
  end
end
Find, $g^* =$ Best solution among the current population
end
return g^*
End

3.6 Grasshopper Optimization Algorithm (GOA)

In the study conducted by Saremi *et al.* [46], a new nature-inspired algorithm named the grasshopper optimization algorithm (GOA) was proposed by mathematically modeling the swarming behavior of grasshoppers in nature. The mathematical model of swarming behavior of grasshoppers include the model for social interaction between grasshoppers. As part of the social interaction, it is considered that the individuals of the grasshopper swarm experience both attractive and repulsive forces as indicated in Fig. 3.3. At a particular distance between two grasshoppers, the attractive and repulsive force is considered to be equal which is known as the comfort zone. At short distances, there is possibility of collision between two grasshoppers. Thus, if the distance between two grasshoppers is less than the distance of comfort zone then the repulsive force between them should be higher in order to avoid collision. On the other hand, in order to form a swarm, the attractive force between two grasshoppers should be higher if the distance between them is more than the comfort zone distance [72]. In each iteration, the best solution is considered as the target for the next iteration which simulates the tendency of grasshoppers to move towards the source of food. While updating the positions of grasshoppers in each iteration, a deceleration coefficient is introduced to gradually obtain a balance between exploration and exploitation while chasing the target solution. The pseudo code for GOA is shown in Algorithm 6 and the steps involved in the optimization process is described as follows.

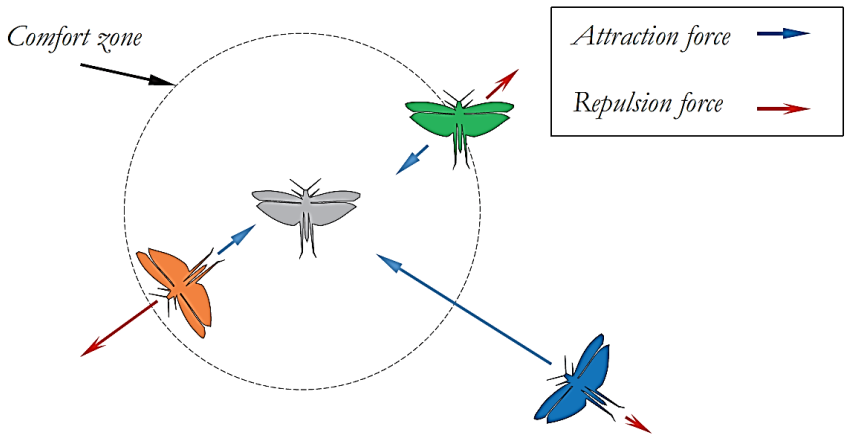


Fig. 3.3: A conceptual model of the interactions between grasshoppers [46]

Step 1: Defining input parameters. (c_{max}, c_{min}, maximum number of iterations)

Step 2: Random initialization of the positions for *n* number of grasshoppers *(X)* within the lower (*lb)* and upper *(ub)* boundaries of the search space.

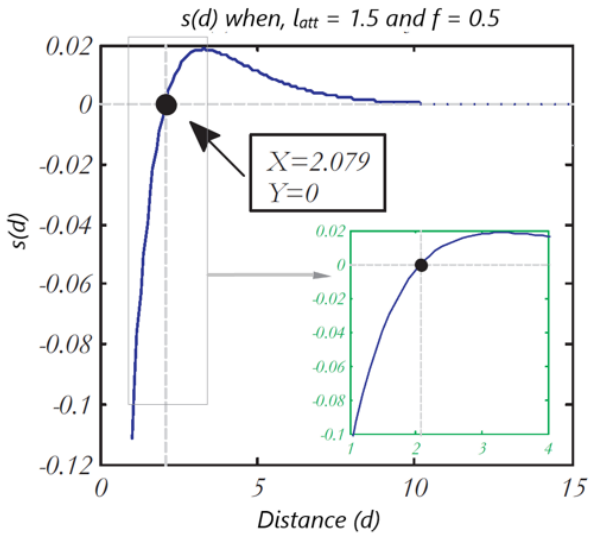


Fig. 3.4: Function ***s*** when $l_{att} = 1.5$ and $f = 0.5$ [46]

Step 3: Evaluation of the fitness values of each search agent and identification of the best one as the target solution (*T*).

Step 4: Calculation of the deceleration coefficient *(c)* based on its maximum (c_{max}) and minimum (c_{min}) values, current iteration number *(l)* and the maximum number of iterations *(L)*.

Step 5: This coefficient *c* is associated in updating the position of each search agent by simulating the social interaction between the grasshoppers through the social interaction function *(s)* and the distance between two grasshoppers (d_{ij}). Fig. 3.4 shows the social interaction function plotted as a function of distance for $l_{att} = 1.5$ and $f = 0.5$. From this figure, it can be seen that the social interaction function *(s)* is most significant within the range from 1 to 4. Thus, the distance between grasshoppers is normalized in the range *[1,4]* [46]. Along with the modeling of social interaction between grasshoppers, the positions of the grasshoppers are moved towards the target solution which imitates the movement of grasshoppers towards the food source. In this way, position of all the *n* number of grasshoppers are updated in this step.

Step 6: The position of the target solution will be updated by any one of the search agents if it obtains a better fitness value. Otherwise, the target solution will remain unchanged.

Step 7: If the stopping criterion is satisfied, then the position of the target solution is considered as the optimum solution. Otherwise, the process will continue from step 4 until the termination criterion is satisfied.

Algorithm 6 Pseudo code for GOA

Begin:
Initialize the positions of the grasshoppers: X_i (i = *1, 2, …, n*)
Initialize c_{max}, c_{min} and maximum number of iterations (L)
Initialize intensity of attraction (f) and the attractive length scale (l_{att})
Evaluate fitness of each search agent: $f(\mathrm{X_i})$ (i = *1, 2, …, n*)
Assign, T = the best search agent
while(the stopping criterion is not satisfied)
Update deceleration coefficient: $c = c_{max} - l\frac{c_{max} - c_{min}}{L}$
for i = 1 to n
Normalize the distance between grasshoppers in *[1,4]*
Update the position of the search agent: $\mathrm{X}_i = c\left(\sum_{\substack{j=1 \\ j\neq i}}^{N} c\frac{ub-lb}{2} s(d_{ij})\frac{x_j - x_i}{d_{ij}}\right) + \hat{T}$
Where, $s(d_{ij}) = fe^{\frac{-d_{ij}}{l_{att}}} - e^{-d_{ij}}$ *and* $d_{ij} = |x_j - x_i|$
Bring the search agent back if it goes outside the boundaries
end
Evaluate fitness of each search agent: $f(\mathrm{X}_i)$ (i = *1, 2, …, n*)
if min($f(\mathrm{X}_i)$) <$f(T)$
Update T with the location of the i^{th} search agent
else
T will remain unchanged
end
l = l+1
end
return T
End

The general optimization characteristics of nature-inspired optimization algorithms are presented in this chapter. The step by step optimization process of some nature-inspired algorithms which are opted for this research are explained with the help of respective pseudocodes. The hybridization technique among these algorithms to form hybrid optimization algorithms for load flow analysis of islanded microgrid is presented in the next chapter.

Chapter 4

Load Flow Analysis and Proposed Hybrid Algorithms

In the conventional load flow analysis, the slack bus governs the voltage and frequency of the whole system. Whereas, in case of droop-controlled islanded microgrid the voltage and frequency are regulated by droop controllers associated with each source. Due to the variation of voltage and frequency, the concept of slack bus becomes invalid in case of islanded microgrid. As a result, for an islanded microgrid with droop-controlled inverter; the system frequency has to be considered as one of the load flow variables along with the voltage magnitudes and reference angles contributed by each inverter in the system. The state variable, *x* can be described in terms of the load flow variables as

$$x = \begin{bmatrix} \omega & \delta_2 & \dots & \dots & \delta_K & v_{oq_1} & \dots & \dots & v_{oq_K} \end{bmatrix} \tag{4.1}$$

where; ω, δ and v_{oq} represents the system frequency, reference angle and voltage magnitude respectively and K represents the total number of inverters in the system. The constraints of the objective problem can be defined as

$$\omega_{min} \le \omega \le \omega_{max}$$
$$\delta_{min} \le \delta \le \delta_{max}$$
$$v_{oq}^{min} \le v_{oq} \le v_{oq}^{max}$$

4.1 Problem Formulation

The objective of the load flow analysis is to minimize the sum of absolute mismatch values of active and reactive power of the inverters. The objective function can be written as

$$Minimize,\ f(x) = \left|\sum_{i=1}^{K} \Delta P_i\right| + \left|\sum_{i=1}^{K} \Delta Q_i\right| \tag{4.2}$$

where; ΔP_i and ΔQ_i are the real and reactive power mismatch at the i^{th} bus. In [48], similar objective function was employed in order to solve the load flow analysis. For a droop-controlled inverter, the power mismatch equations are the difference between the inverter output power

calculated at the global reference frame and the reference values set by the droop controllers. For the i^{th} inverter bus the active and reactive power mismatch equations are

$$\Delta P_i = \frac{3}{2}(V_{od_i} I_{od_i} + V_{oq_i} I_{oq_i}) - \frac{(\omega_n - \omega)}{m_i} \tag{4.3}$$

$$\Delta Q_i = \frac{3}{2}(V_{oq_i} I_{od_i} - V_{od_i} I_{oq_i}) - \frac{(V_n - v_{oq_i})}{n_i} \tag{4.4}$$

To determine the power mismatch values, a set of equations has to be solved which involve the calculation of bus voltages and inverter output currents. For these calculations an equivalent circuit of the inverter model is considered as shown in Fig. 4.1, where the inverter output voltage across the capacitor is considered as a voltage source behind its coupling impedance. The process of determining the power mismatch values is described in the following steps.

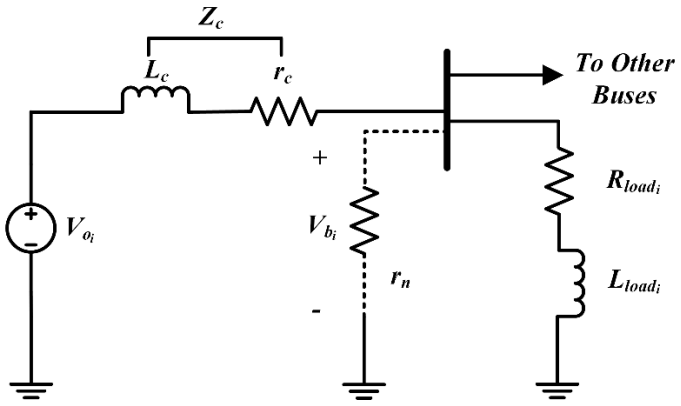


Fig. 4.1: Steady-state equivalent circuit of inverter model at bus *i*

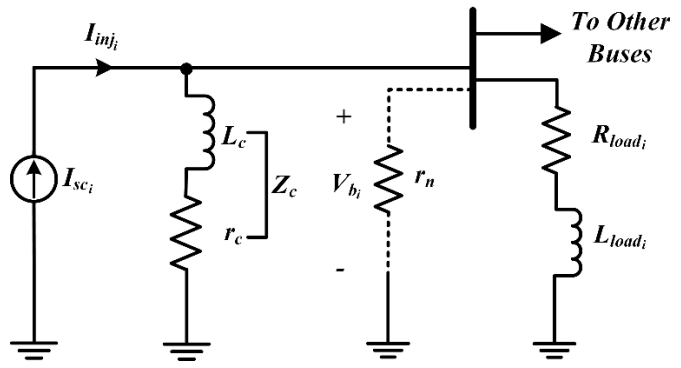


Fig. 4.2: Norton equivalent circuit of steady-state inverter model at bus *i*

Step 1: First of all, following equations (2.27) and (2.28); the *d-axis* and *q-axis* components of the inverter output voltage is transformed in the global reference frame using the reference angle δ. Then, the output voltage of the i^{th} inverter in terms of a complex quantity can be calculated as

$$V_{od_i} = sin(\delta_i)v_{oq_i} \tag{4.5}$$

$$V_{oq_i} = cos(\delta_i)v_{oq_i} \tag{4.6}$$

$$V_{o_i} = V_{od_i} + jV_{oq_i} \tag{4.7}$$

Step 2: Before determining the bus voltages it is required to calculate the current injected to a particular bus. For this study, only constant impedance loads are considered. So, the inverter are the only sources to inject current to their respective buses. The current injected by the inverters can be easily calculated by transforming the circuit shown in Fig. 4.1 to its Norton equivalent as shown in Fig. 4.2. Then, the current injected by the i^{th} inverter is calculated as

$$I_{inj_i} = I_{SC_i} = \frac{V_{o_i}}{Z_c(\omega)} \tag{4.8}$$

Step 3: The bus voltages can now be calculated from the injected currents as

$$V_b = Z_{bus}(\omega)I_{inj} \tag{4.9}$$

For islanded microgrids the bus impedance matrix is a function of frequency and it has to be updated at each iteration. For an *N*-bus system the vector of injected currents at each bus is given by

$$I_{inj} = \begin{cases} I_{inj_p} & due\ to\ inverter\ at\ bus\ p\ (p \leq N) \\ 0 & otherwise \end{cases} \tag{4.10}$$

Step 4: After determining the bus voltages, the output current of the i^{th} inverter can be determined by

$$I_{o_i} = \frac{V_{o_i} - V_{b_i}}{Z_c(\omega)} \tag{4.11}$$

The *d-axis* and *q-axis* components of the inverter output current in the global reference frame is given by

$$I_{od_i} = \Re\{I_{o_i}\} \tag{4.12}$$

$$I_{oq_i} = \Im\{I_{o_i}\} \tag{4.13}$$

Step 5: The equations from (4.5) to (4.13) are sufficient to calculate the active and reactive power mismatch values for each inverter by solving the equations described in (4.3) and (4.4).

Thus, the values of the power mismatch equations can be used to evaluate the objective function as indicated in equation (4.2).

4.2 Proposed Hybrid Algorithms

The flowcharts of the hybrid algorithms are shown in Fig. 4.3 and Fig. 4.4. Several hybrid algorithms based on nature-inspired optimization algorithms have been proposed to perform the load flow analysis of islanded microgrids. These hybrid algorithms have been designed keeping the imperialist competitive algorithm (ICA) as the main frame. Four other metaheuristic algorithms namely; genetic algorithm (GA), differential evolution (DE), flower pollination algorithm (FPA) and grasshopper optimization algorithm (GOA) were separately combined with ICA to obtain four hybrid algorithms namely ICGA, ICDE, ICFPA and ICGOA. The main focus of introducing GA, DE, FPA and GOA in the optimization process of ICA is to enhance the capability of better exploration and exploitation of the search space. Exploration and exploitation are two key terms of these nature-inspired optimization algorithms. Where, exploration refers to a process of searching the search space in a wider range to increase the diversity of candidate solutions. On the other hand, exploitation indicates the process of searching in the vicinity of prospective solutions. Thus, exploration can be regarded as a search in the global scale, whereas exploitation indicates search in the local scale. Adding other nature inspired algorithms along with ICA will introduce few more steps to update the candidate solutions in the optimization process which is expected to increase the diversity among the generated solution sets. Furthermore, if a situation arises where the optimization problem is stuck in a local minima, then addition of GA, DE, FPA or GOA in the optimization process of ICA may assist in getting out of the local minima and approach towards convergence faster. The process of applying the proposed hybrid algorithms for the load flow analysis is described in the following steps.

Step 1: Initialization of the system data of islanded microgrid.

Step 2: Generation of initial population for the state variables as stated in (4.1). In case of ICA, the population individuals are called countries. For a particular study a specific number of countries are initially generated.

Step 3: For each country, equations (4.5) to (4.13) are solved and the values of active and reactive power mismatch for each inverter are calculated using equations (4.3) and (4.4). Then, for each country in the population the value of the objective function is determined using equation (4.2).

Step 4: Next, the countries are sorted according to their objective function values. Then, depending on the fitness values of the countries; the empires are generated by setting specific countries as the imperialists and assigning rest of the countries as colonies to them.

Step 5: The positions of the colonies are then moved towards the position of their respective imperialist by a process called assimilation.

Step 6: In this step, the positions of some of the colonies are modified randomly by doing revolution.

Step 7: If there is a colony which has a lower fitness value than the imperialist, then their positions are interchanged. This process is referred as intra-empire competition.

Step 8: The hybridization process of ICA with GA, DE, FPA and GOA is carried out in this step. Four distinct hybrid algorithms are obtained by following the four cases as indicated below

Case 1: The flowchart of ICGA is shown in Fig. 4.3(a), where the updated empires from the previous step is set as parents in GA. Then, the positions of the imperialists and colonies are updated through crossover, mutation and selection process.

Case 2: The resulting algorithm will be ICDE if the flowchart shown in Fig. 4.3(b) is followed. Here, the empires are assigned as parents of the DE algorithm. The positions of the countries (imperialists and colonies) are updated through mutation, crossover and selection process of DE.

Case 3: In Fig. 4.4(a), the flowchart of ICFPA is presented. The empires are assigned as population of flowers in FPA. The population is then modified by mimicking either global pollination or local pollination process depending upon a probability switch.

Case 4: Hybrid algorithm ICGOA can be obtained by following the algorithm shown in Fig. 4.4(b). In this case the empires are set as the positions of grasshoppers in GOA. The positions are updated by simulating the swarming behavior of grasshoppers.

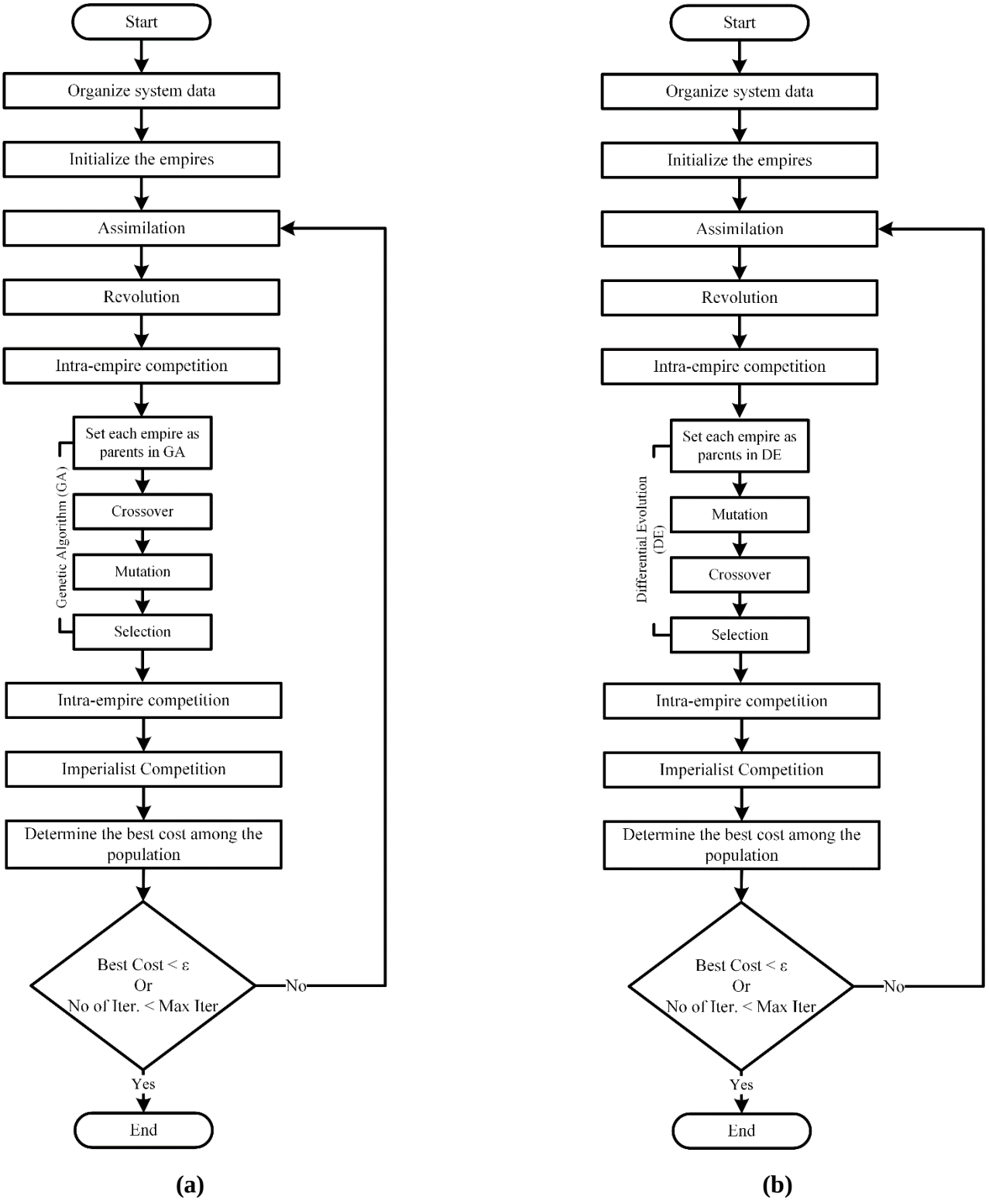


Fig. 4.3: Flowchart of; **(a)** ICGA and **(b)** ICDE

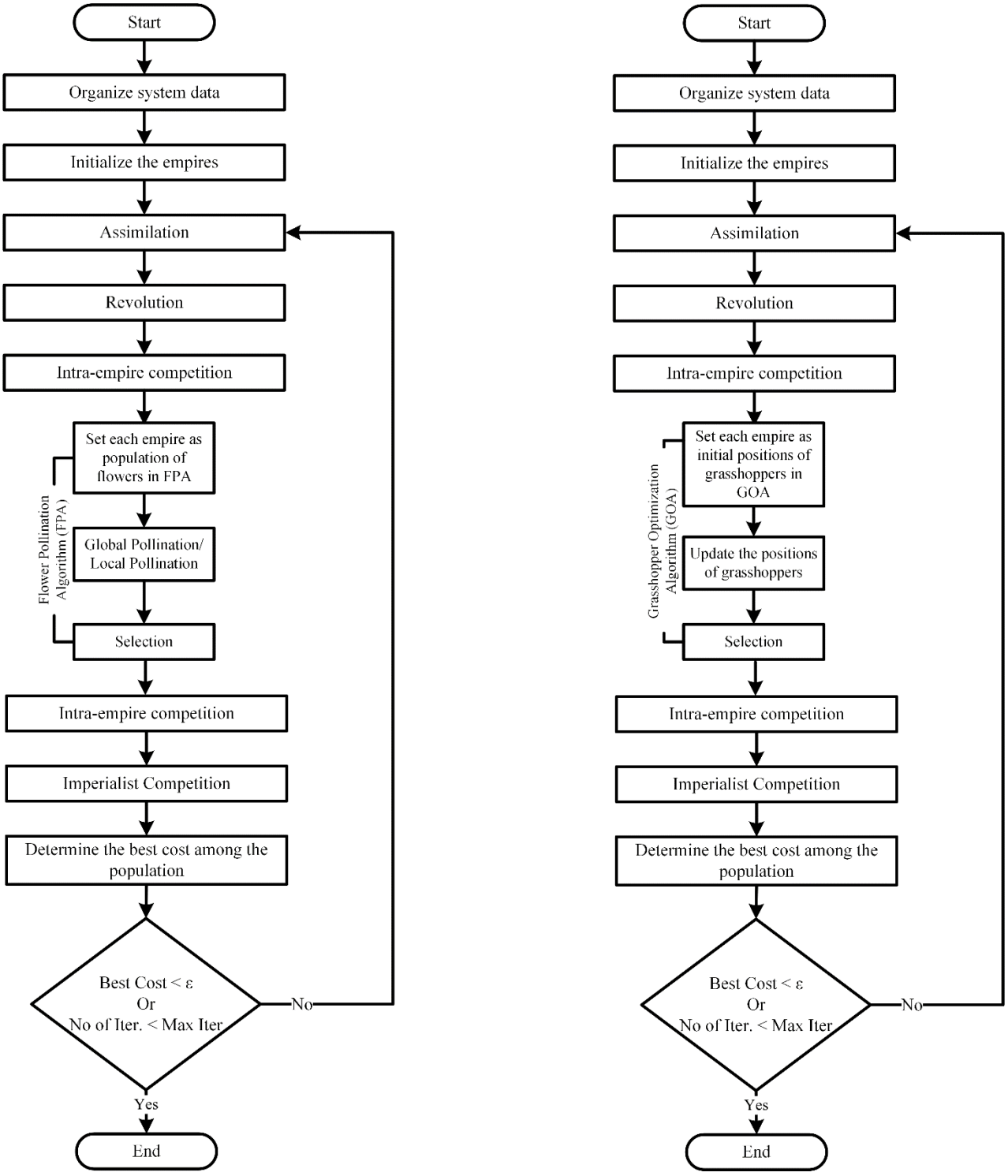


Fig. 4.4: Flowchart of; **(a)** ICFPA and **(b)** ICGOA

Selecting any one of these four cases in this step is the prime difference among the four hybrid algorithms. The rest of the steps are similar for each algorithm.

Step 9: Intra-empire competition is performed again as mentioned in step 7.

Step 10: In this step, first of all the total fitness of each empire is calculated. Then, in imperialistic competition, the weakest colonies are identified and are given to the empires which have the most likelihood to possess them. If an empire ends up with no colonies then that empire will be eliminated.

Step 11: The solution set which is providing the best fitness value will be identified in this step.

Step 12: If the stopping conditions are satisfied, then the whole process will be terminated. Other-wise, the calculations will be repeated from step 5. For this study, the optimization process will terminate if any one of the following two stopping criterion is satisfied.

1. If the value of the best fitness is less than a pre-specified threshold (ε) value.
2. If the total number of iterations is less than a pre-specified value of maximum number of iterations.

In this chapter, the load flow variables for an islanded microgrid were defined with respect to the mathematical model described in Chapter 2 and an objective function was formulated in terms of the load flow variables to perform the load flow analysis as an optimization problem. In the later portion of the chapter, discussion was carried out regarding the hybridization technique employed in this study to obtain different hybrid algorithms. Applying the proposed hybrid algorithms to perform the load flow analysis of a case study system is presented in the next chapter.

Chapter 5

Numerical Case Study

In order to perform load flow analysis using the hybrid algorithms presented in the previous chapter the modified IEEE 37-bus system is considered as the case study system. Along with these hybrid algorithms, the load flow analysis of the same case study system is performed using PSO. Based on the performance of each algorithm in solving the load flow problem, a comparative study among these algorithms will be presented in this chapter. Before going to the comparative study, this chapter starts with a brief discussion on the modified IEEE 37-bus system. Then, the algorithm with the better performance is identified through the comparative study and the corresponding load flow results are demonstrated in the later portion of the chapter.

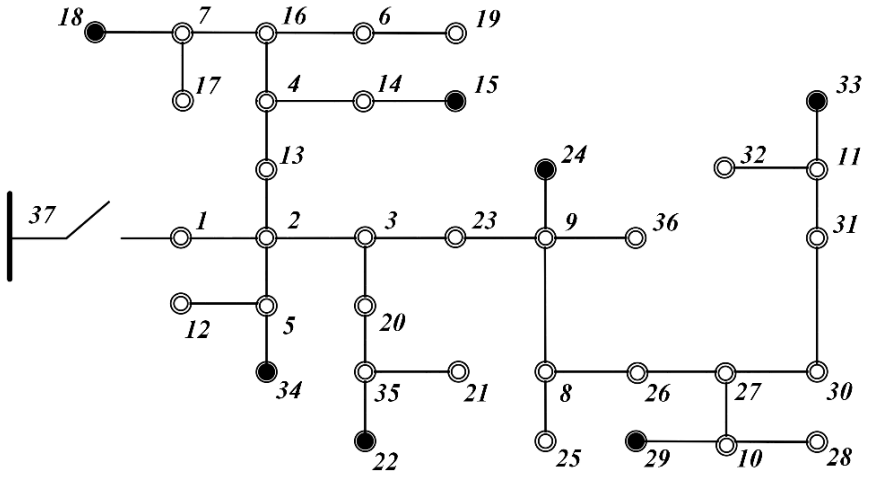


Fig. 5.1: Single-line diagram of the modified IEEE 37-bus system

5.1 System Information

The standard IEEE 37-bus system is modified by connecting seven inverters at different bus locations as indicated in [73]. The single line diagram of the modified IEEE 37-bus system is shown in Fig. 5.1. The seven inverters are connected at buses 15, 18, 22, 24, 29, 33 and 34 as indicated by the black dots in Fig. 5.1. For this modified IEEE 37-bus system load flow analysis was performed using a quasi-Newton method in [21]. In order to conduct a comparative study, the different parameters associated with the load flow analysis for this IEEE 37-bus system were

taken similar to the ones in [21]. For all the inverters, a nominal voltage of V_n = 170 V was chosen and nominal frequency was set to $\omega_n = 2\pi 60$ *rad/s*. The maximum power ratings and the droop co-efficients for each inverter is given in Table 5.1. Only constant impedance loads are considered in this case study. The branch and load parameters for the IEEE 37-bus system are presented in Table 5.2 and Table 5.3.

Table 5.1: Inverter bus locations, power ratings and droop co-efficients [21]

i	*Bus*	P_{max} *(kW)*	Q_{max} *(kVAR)*	m_i^{-1}	n_i^{-1}
1	15	15	15	2387.3	1250
2	18	8	8	1273.2	666.7
3	22	10	10	1591.5	833.3
4	24	15	15	2387.3	1250
5	29	8	8	1273.2	666.7
6	33	10	10	1591.5	833.3
7	34	15	15	2387.3	1250

Table 5.2: Branch Parameters [73]

From Bus	To Bus	R (Ω)	L (H)	From Bus	To Bus	R (Ω)	L (H)
1	2	0.167	2.31×10^{-4}	10	29	0.223	3.08×10^{-4}
2	5	0.070	9.64×10^{-5}	11	33	0.070	9.64×10^{-5}
2	13	0.063	8.67×10^{-5}	11	32	0.035	4.82×10^{-5}
2	3	0.230	3.18×10^{-4}	13	4	0.091	1.25×10^{-4}
3	20	0.042	5.78×10^{-5}	14	15	0.091	1.25×10^{-4}
3	23	0.105	1.45×10^{-4}	16	7	0.160	2.22×10^{-4}
4	14	0.014	1.93×10^{-5}	16	6	0.105	1.45×10^{-4}
4	16	0.139	1.93×10^{-4}	20	35	0.049	6.75×10^{-5}
5	34	0.056	7.71×10^{-5}	23	9	0.035	4.82×10^{-5}
5	12	0.042	5.78×10^{-5}	26	27	0.098	1.35×10^{-4}
6	19	0.049	6.75×10^{-5}	27	30	0.112	1.54×10^{-4}
7	18	0.132	1.83×10^{-4}	27	10	0.091	1.25×10^{-4}
7	17	0.021	2.89×10^{-5}	30	31	0.070	9.64×10^{-5}
8	26	0.056	7.71×10^{-5}	31	11	0.070	9.64×10^{-5}
8	25	0.056	7.71×10^{-5}	35	21	0.035	4.82×10^{-5}
9	24	0.105	1.45×10^{-4}	35	22	0.049	6.75×10^{-5}
9	8	0.056	7.71×10^{-5}	36	9	0.230	3.18×10^{-4}
10	28	0.035	4.82×10^{-5}				

Table 5.3: Load Parameters [73]

Bus	R (Ω)	L (H)	Bus	R (Ω)	L (H)
1	6.58	0.0105	24	49.93	0.0748
12	49.93	0.0748	25	98.74	0.1572
13	49.93	0.0748	26	49.93	0.0748
14	111.41	0.1681	27	98.74	0.1572
15	49.93	0.0748	28	49.93	0.0748
16	49.93	0.0748	29	98.74	0.1572
17	25.82	0.0409	30	29.62	0.0472
18	98.74	0.1572	31	33.12	0.0519
19	98.74	0.1572	32	49.93	0.0748
20	98.74	0.1572	33	98.74	0.1572
21	32.91	0.0524	34	45.54	0.0686
22	98.74	0.1572	35	98.74	0.1572
23	49.93	0.0748			

For the modified IEEE 37-bus system the load flow analysis was formulated as an optimization problem as discussed in chapter 4. Next, load flow analysis was performed by applying the hybrid algorithms ICGA, ICDE, ICFPA and ICGOA separately to the modified IEEE 37-bus system. For these hybrid algorithms, the total number of countries was set to 100. For this study, among these 100 countries, 5 were chosen as imperialists and rest of them were assigned as colonies to the imperialists. Later on, PSO was also applied to perform the load flow analysis of the same case study system. The total number of particles for PSO was set to 100 as well, in order to make a logical comparison among the five algorithms. The values different parameters of these algorithms used for the simulation is summarized in Table 5.4 and Table 5.5. All the simulations were performed using a personal computer with a processor of intel core *i7-8550* at 1.8 *GHz* and with an installed RAM of 8 *GB*. For this case study, the following stopping criterion were considered which would terminate the optimization process once satisfied.

1. If the best fitness value is less than a pre-specified threshold (ε) value which was set to 10^{-5} for this study.
2. If the number of iterations is equal to a pre-specified value of maximum number of iterations. For this study, the maximum number of iterations was set to 100.

Table 5.4: Parameters of ICGA, ICDE, ICFPA, ICGOA

Algorithm	Parameters	Algorithm	Parameters
ICA	$n_{pop} = 100$ $n_{imp} = 5$ $n_{col} = 95$ $\beta = 1.5$ $p_R = 0.05$ $\mu = 0.1$ $\xi = 0.2$	GA	$p_c = 0.7$ $p_m = 0.3$ $\mu_m = 0.1$ $\gamma = 0.4$
		DE	$p_{CR} = 0.5$ $F \sim [0.4,1]$
		FPA	$p = 0.8$
		GOA	$c_{max} = 1$ $c_{min} = 0.00004$ $f = 0.5$ $l_{att} = 1.5$

Table 5.5: Parameters of PSO

Algorithm	Parameters
PSO	$n = 100$ $w_{damp} = 0.99$ $C_1 = 2.05$ $C_2 = 2.05$

5.2 Comparison among the Algorithms with Statistical Analysis

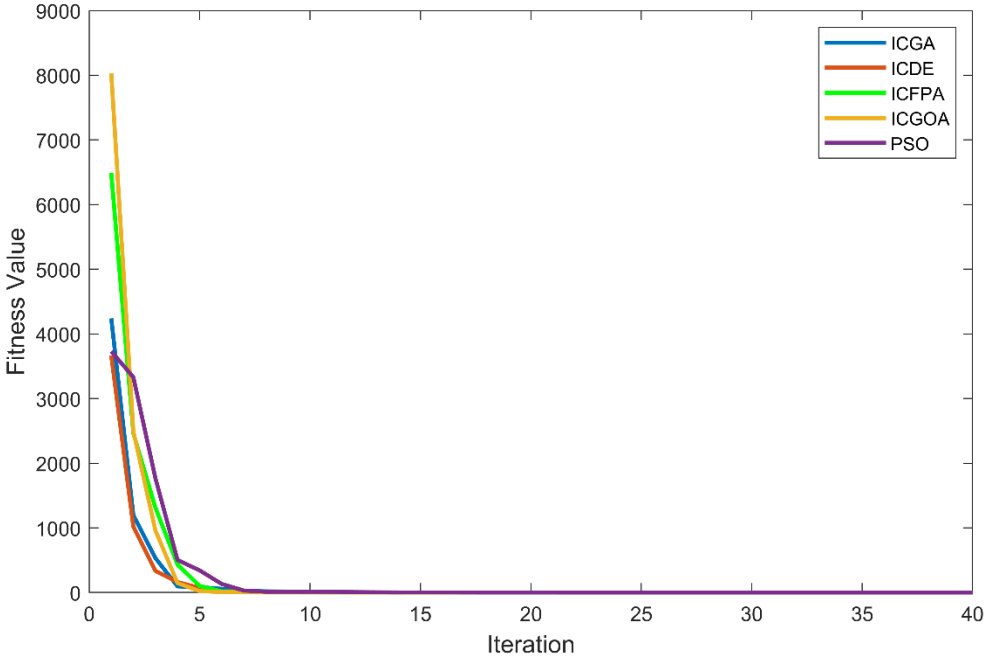
Being metaheuristic algorithms, these algorithms operate in a stochastic manner which indicates that the optimization process is associated with randomness. The optimization process for these algorithms start by randomly initializing the positions of the solution variables within the boundary of pre-specified constraints. Then, a number of iterations is performed by updating the positions of each solution set through a series of random process until the optimum solution is obtained. Due to the inherent randomness of these algorithms it is most likely that the number of iterations and the execution time needed to complete the optimization process may vary for each independent run/trial. Thus, for overall comparison among the algorithms, each algorithm was executed for 30 independent trials.

For each independent trial, the number of iterations to reach the stopping criterion and the overall execution time were recorded. These data are summarized in Table 5.6 in three categories namely best, average and worst result for each algorithm. From Table 5.6, among the 30 independent runs if the best results are considered, then it can be observed that ICGA and ICDE achieve convergence in minimum number of iterations which is 23 in this case. For this best-case

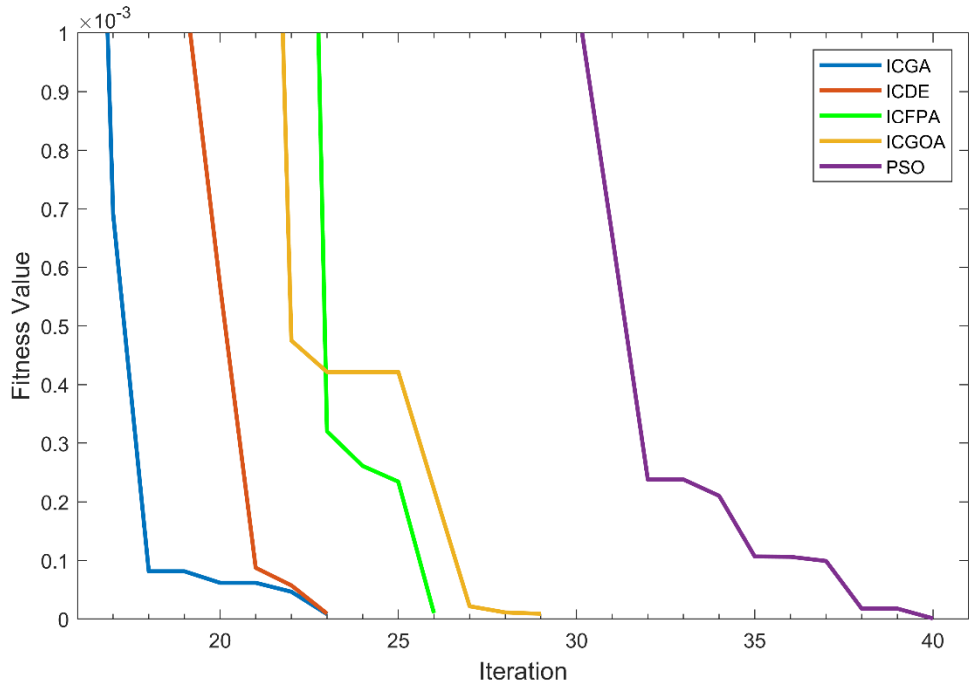
Table 5.6: Iterations and Time required by ICGA, ICDE, ICFPA, ICGOA and PSO algorithms

Algorithm	Best Results		Average Results		Worst Results	
	No. of Iterations	Execution Time (sec)	No. of Iterations	Execution Time (sec)	No. of Iterations	Execution Time (sec)
ICGA	23	80.215	30	105.528	40	140.287
ICDE	23	79.659	27	95.512	30	120.108
ICFPA	26	90.597	32	146.213	39	305.142
ICGOA	29	101.586	42	159.265	50	569.403
PSO	40	67.277	55	96.175	81	143.393

scenario, if the execution time is taken into consideration then it can be seen that ICDE completes the optimization process 0.556 *sec* faster compared to ICGA. In case of ICFPA and ICGOA the required number of iterations and the execution time is higher compared to the other algorithms. For PSO, 40 iterations were required to complete the load flow analysis; however, due to less complexity in the optimization process, convergence is achieved faster compared to the other algorithms. On the other hand, for the worst-case scenario, ICDE completes the load flow analysis in 30 iteration with a computational time of 120.108 *sec*. Whereas, the other algorithms require more iterations and higher computational time to complete the load flow analysis. To summarize the results shown in Table 5.6, considering all 30 independent trials, on an average, ICDE requires 27 iterations to complete the optimization process with an average computational time of 95.512 *sec*. In case of PSO the average computational time is 96.175 *sec* which is close to that of ICDE, but the required number of iterations is much higher compared to ICDE. Thus, among the five algorithms ICDE can be considered as the algorithm with better performance. To further support the above discussion, the convergence graph of each algorithm considering their best and worst results is respectively shown in Fig. 5.2 and Fig. 5.3. From Fig. 5.2(a) and Fig. 5.3(a), it can be seen that all the algorithms attain very high fitness values in the initial iterations and gradually after completing several iterations the fitness values obtain convergence. In order to show the exact point of convergence for each algorithm, the zoomed version of the graphs shown in Fig. 5.2(a) and Fig. 5.3(a) are depicted in Fig. 5.2(b) and Fig. 5.3(b). From these two figures, it is also evident that for both the cases; ICDE is providing faster convergence compared to the other algorithms. For the 30 independent trials, the number of iterations required in each trial for the five individual algorithms is shown in Fig. 5.4.

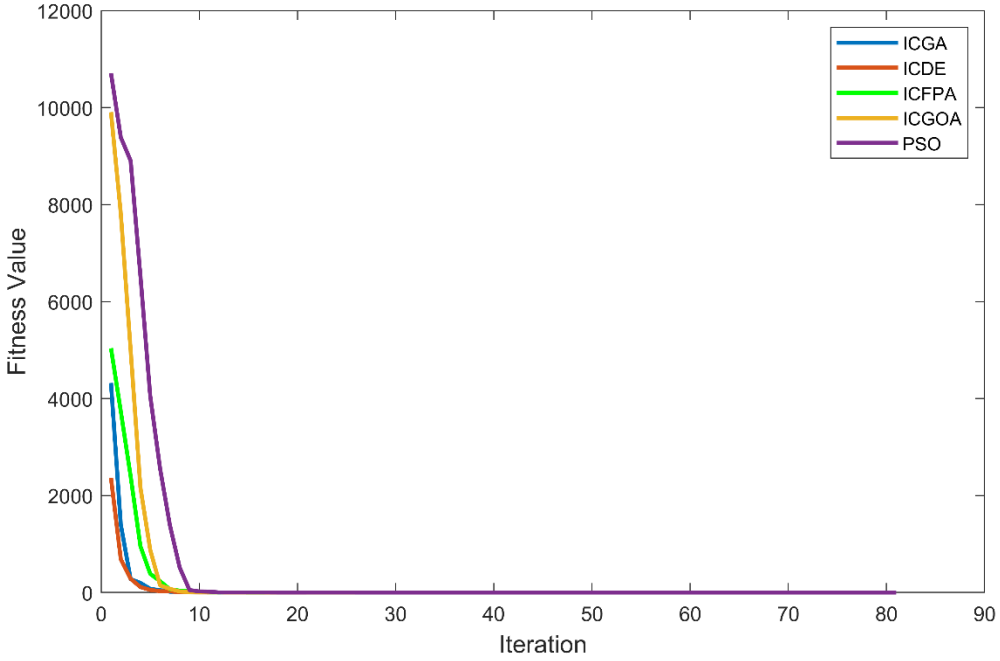


(a)

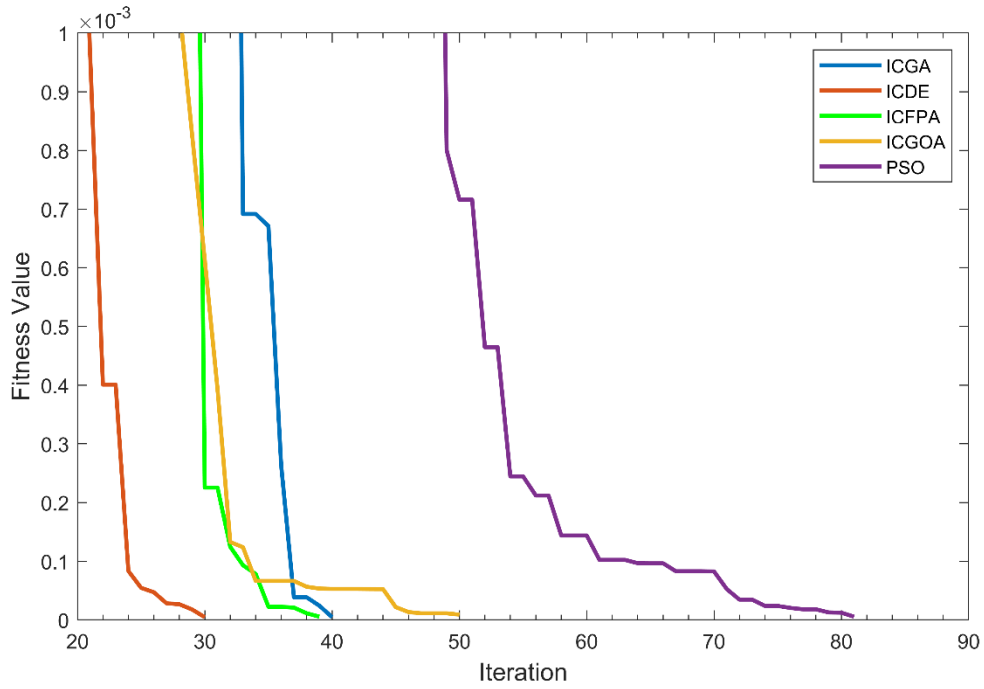


(b)

Fig. 5.2: Convergence graph for the best results of each algorithm; **(a)** original scale**, (b)** zoomed version



(a)



(b)

Fig. 5.3: Convergence graph for the worst results of each algorithm; **(a)** original scale**,** **(b)** zoomed version

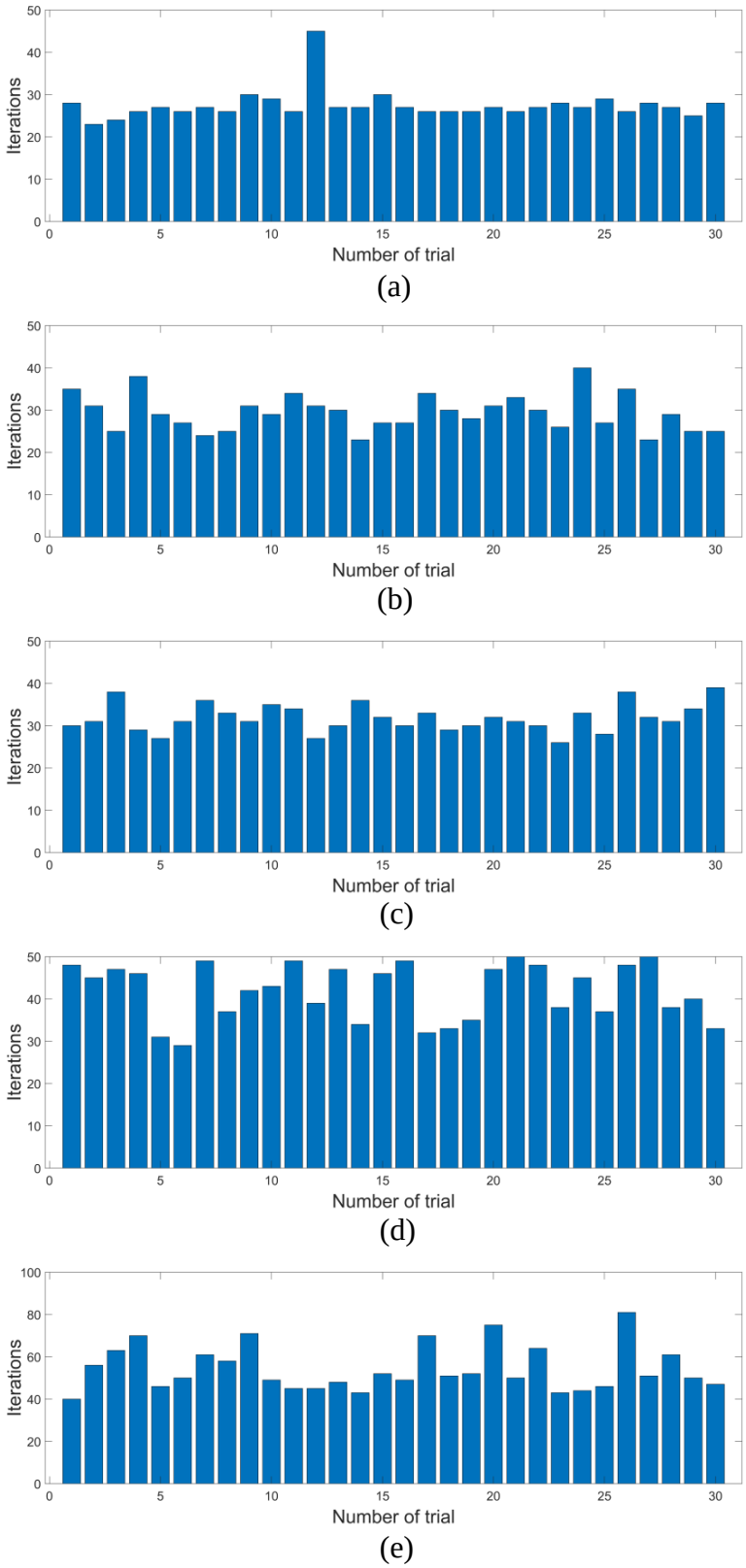


Fig. 5.4: Total number of iterations in each trial in case of; **(a)** ICDE, **(b)** ICGA, **(c)** ICFPA, **(d)** ICGOA, **(e)** PSO

From the bar plots shown in Fig. 5.4, it can be observed that ICDE has more uniform distribution compared to the other algorithms which indicates that ICDE has less variation in the required number of iterations for different trials.

For further analysis of the acquired results, SPSS statistics software was used to perform statistical analysis of the obtained data from 30 independent runs. To demonstrate the uniqueness of each algorithm, independent samples *t-tests* were performed to compare the means of each algorithm. In this study, data from two algorithms were defined as grouping variables at a time. Table 5.7 and Table 5.8 show the *t-test* results based on the required number of iterations and the execution time as the comparison variable respectively. Whenever independent samples *t-test* is performed in SPSS, the software also generates results of a corresponding *F-test* which determine whether the data sample of two groups have equal variances or not. If the significant factor (*p-value*) of the *F-test* is greater than the significance level of 0.05, then the group variances are considered to be equal. Otherwise, it is not possible to assume equal variances. From Table 5.7, it can be observed that when the number of iterations is considered, data samples of the pairs ICFPA-ICGA and ICGOA-PSO possess equal variances as the *p-value* of the F-test is greater than 0.05. All the other pair of groups possess non-equal variances in this case. Similarly, as shown in Table 5.8, in terms of the computational time, the pairs which exhibit equal variances are ICFPA-ICGOA, ICGA-ICGOA, ICGA-PSO and ICGOA-PSO.

For the *t-test*, the null hypothesis H_0 assumes that the mean values of the data set are equal and the alternative hypothesis H_1 assumes that the mean values of the data set are not equal. Whether the null hypothesis can be accepted or not depends on the value of the significant factor (*p-value*) of the *t-test.* From Table 5.7, it can be seen that, the *p-value* of the *t-test* with respect to the required number of iterations is smaller than 0.05 for all the pairs of data sample, which indicate that the null hypothesis can be rejected and in this context there is significant difference among all the algorithms. On the other hand, Table 5.8 shows the *t-test* results with respect to computational time. Considering the *p-values* of the *t-test*s shown in Table 5.8, it can be observed that all pairs except ICDE-PSO and ICFPA-ICGOA contain significant difference with respect to execution time.

Thus, considering both execution time and the number of iterations required to complete the optimization process, all the algorithms can be regarded to have unique characteristics. Based on the studies done so far, it can be observed that ICDE performs relatively better compared to the other algorithms considering the average number of iterations and the average execution time.

Table 5.7: Results of t-test based on number of Iterations

Methods	***F*-test**		***t*-test for equality of means**			
	F	**Sig.**	**Mean Difference**	**t**	**df**	**Sig. (2-tailed)**
ICDE-ICFPA	12.819	0.001	-5.033	-7.560	40.905	0.000
ICDE-ICGA	21.065	0.000	-2.567	-3.067	36.191	0.004
ICDE-ICGOA	77.330	0.000	-15.033	-10.687	31.387	0.000
ICDE-PSO	50.681	0.000	-27.533	-13.987	30.196	0.000
ICFPA-ICGA	1.992	0.164	2.467	2.483	58	0.016
ICFPA-ICGOA	33.548	0.000	-10.000	-6.644	39.740	0.000
ICFPA-PSO	30.552	0.000	-22.5	-11.030	34.524	0.000
ICGA-ICGOA	18.166	0.000	-12.467	-7.849	46.147	0.000
ICGA-PSO	21.454	0.000	-24.967	-11.877	38.255	0.000
ICGOA-PSO	3.020	0.088	-12.5	-5.237	58	0.000

Table 5.8: Results of t-test based on computational time

Methods	***F*-test**		***t*-test for equality of means**			
	F	**Sig.**	**Mean Difference**	**t**	**df**	**Sig. (2-tailed)**
ICDE-ICFPA	38.033	0.000	-50.701	-3.941	29.805	0.000
ICDE-ICGA	8.777	0.004	-10.016	-3.137	44.320	0.003
ICDE-ICGOA	5.209	0.026	-63.754	-4.259	29.592	0.000
ICDE-PSO	17.358	0.000	-0.664	-0.175	39.528	0.862
ICFPA-ICGA	28.486	0.000	40.685	3.110	31.810	0.004
ICFPA-ICGOA	1.375	0.246	-13.052	-0.665	58	0.509
ICFPA-PSO	23.825	0.000	50.037	3.779	33.259	0.001
ICGA-ICGOA	3.360	0.072	-53.738	-3.545	58	0.001
ICGA-PSO	1.629	0.207	9.352	2.092	58	0.041
ICGOA-PSO	2.493	0.120	63.089	4.125	58	0.000

Table 5.9: Load flow results obtained by ICDE

Load Flow Variable	Calculated Value	Load Flow Variable	Calculated Value
ω *(rad/s)*	375.5576	$v_{oq_1}(V)$	165.472
δ_2 *(deg)*	-1.3682	$v_{oq_2}(V)$	168.236
δ_3 *(deg)*	-2.9203	$v_{oq_3}(V)$	169.621
δ_4 *(deg)*	-0.3786	$v_{oq_4}(V)$	169.717
δ_5 *(deg)*	1.0963	$v_{oq_5}(V)$	166.986
δ_6 *(deg)*	2.2311	$v_{oq_6}(V)$	169.932
δ_7 *(deg)*	2.4701	$v_{oq_7}(V)$	168.741

Table 5.10: Comparison among the per unit output voltages at each inverter obtained through ICDE and quasi-Newton method

i	**Bus**	**Output Voltage (p.u.)**	
		ICDE	**Quasi-Newton [21]**
1	15	0.9734	0.9789
2	18	0.9896	0.9601
3	22	0.9978	0.9655
4	24	0.9983	0.9844
5	29	0.9823	0.9745
6	33	0.9996	0.9673
7	34	0.9926	0.9700

Table 5.11: Generated active and reactive powers at each inverter in case of ICDE

i	Bus	Generated Power	
		Active Power (kW)	**Reactive Power (kVAR)**
1	15	1.856	0.685
2	18	3.477	0.357
3	22	6.595	-1.424
4	24	3.235	1.330
5	29	0.905	0.803
6	33	1.685	3.493
7	34	0.729	5.899

5.3 Results of Load Flow Analysis

The results of the load flow analysis of the modified IEEE 37-bus system using ICDE is given in Table 5.9. In this table the optimized values of the load flow variables such as the system frequency, reference frame angles and the *q-axis* component of the inverter output voltages are recorded. Among the 30 independent runs the best result is tabulated here. From the obtained results the (p.u.) value of the steady state frequency is found to be 0.9962. In [21], the load flow solution of the modified IEEE 37-bus system was obtained through a quasi-Newton method. In order to make a comparison among the ICDE and the quasi-Newton method, the per unit (p.u.) values of the inverter output voltages at each inverter bus is tabulated in Table 5.10. From Table 5.10, it can be observed that the p.u. values of the inverter output voltages obtained through both the algorithms are very close to each other and it can also be seen that all bus voltages lie within 5% of the rated bus voltage which satisfy the IEEE standard of voltage regulation as stated in [74]. Furthermore, it can be observed that the p.u. values of the inverter output voltages are close to unity in case of ICDE which indicate that the voltages in this case are close to the nominal value. The inverter bus locations and the values of the active and reactive powers generated by each inverter in case of ICDE is tabulated in Table 5.11.

The results obtained so far are sufficient to calculate the voltages and phase angles at other buses of the network. These information are important for proper monitoring and operation of the whole system. The obtained load flow results can also be used to calculate steady-state operating points that can be used to linearize the nonlinear equations of the system model which is necessary for control and small signal stability analysis of the system.

As ICDE was found to exhibit better performance compared to the other algorithms considered in this work, the MATLAB code of ICDE is presented in Appendix. The resources available in [75] were helpful in developing the MATLAB codes for the implementation of the hybrid algorithms.

Chapter 6

Conclusion and Future Research Directions

6.1 Conclusion

This thesis was focused on the application of nature-inspired hybrid optimization algorithms for the efficient solution of load flow problem of autonomous microgrids. In **Chapter 1**, brief introduction to microgrid systems, its different operating modes, significance of droop control scheme for autonomous microgrids and importance of load flow analysis were presented. Motivation towards this research and the thesis objectives were outlined in the later portion of the chapter in light of the literature review. **Chapter 2** contains the mathematical model of autonomous microgrid in synchronous reference frame. The mathematical is useful in understanding the relationship between different parameters associated with the microgrid system. In **Chapter 3**, a brief discussion was presented on the general characteristics of nature inspired optimization techniques along with the optimization process of some of the algorithms considered for this study. For solving the load flow problem, an objective function was formulated based on the absolute summation of errors in the real and reactive power generations from the inverter based microgrid sources as demonstrated in **Chapter 4**. This chapter also contains discussions on the formation of hybrid optimization techniques; namely ICGA, ICDE, ICFPA and ICGOA. The hybridization was performed with a view to improving the global searching capability by enhanced exploration of the search space. In **Chapter 5**, the hybrid algorithms were applied to conduct a case study on the modified IEEE 37-bus system containing seven droop-controlled inverters. The simulations for the load flow analysis were carried out using MATLAB software. For the same case study system, PSO was also applied to conduct the load flow analysis. The performances of the applied hybrid algorithms along with PSO were compared through a series of statistical tests. SPSS statistical analysis software was used to conduct the statistical analysis. Based on the statistical test ICDE was found to exhibit better performance than the other algorithms in terms of the required number of iterations and the execution time. Therefore, ICDE can be regarded as a prospective stochastic load flow technique for droop-controlled islanded microgrids.

6.2 Future Research Directions

With respect to the study conducted in this research some possible future research scopes can be suggested. Apart from the algorithms considered in this thesis, other nature-inspired metaheuristic algorithms may be adopted for the load flow analysis of islanded microgirds and a comparative study may be conducted. Furthermore, in this thesis, the objective function was considered to be the absolute summation of the real and reactive power mismatch. So, the objective function actually does not consider the mismatch of real and reactive power at each bus, rather the mismatch of all the inverter buses were considered collectively. As an alternative approach, the load flow problem can be considered as a multi-objective optimization problem where the power mismatch equations at each bus can be considered as separate objective functions. Then, with the help of multi-objective optimization technique a particular solution point has to be identified where all the separate objective function will meet the specified convergence criteria.

References

[1] R. H. Lasseter and P. Piagi, "Microgrid: A conceptual solution," in *IEEE Power Electronics Specialists Conference*, 2004, pp. 4285-4291.

[2] N. Hatziargyriou, *Microgrids: architectures and control*: John Wiley & Sons, 2014.

[3] R. Lasseter, A. Akhil, C. Marnay, J. Stephens, J. Dagle, R. Guttromsom, *et al.*, "Integration of distributed energy resources. The CERTS Microgrid Concept," Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States)2002.

[4] R. Lasseter and P. Piagi, "Providing premium power through distributed resources," in *Proceedings of the 33rd annual Hawaii international conference on system sciences*, 2000, p. 9 pp.

[5] B. Hartono and R. Setiabudy, "Review of microgrid technology," in *2013 international conference on QiR*, 2013, pp. 127-132.

[6] F. Katiraei and M. R. Iravani, "Power management strategies for a microgrid with multiple distributed generation units," *IEEE transactions on power systems*, vol. 21, pp. 1821-1831, 2006.

[7] T. L. Vandoorn, J. C. Vasquez, J. De Kooning, J. M. Guerrero, and L. Vandevelde, "Microgrids: Hierarchical control and an overview of the control and reserve management strategies," *IEEE industrial electronics magazine*, vol. 7, pp. 42-55, 2013.

[8] Z. Shuai, Y. Sun, Z. J. Shen, W. Tian, C. Tu, Y. Li, *et al.*, "Microgrid stability: Classification and a review," *Renewable and Sustainable Energy Reviews*, vol. 58, pp. 167-179, 2016.

[9] J. P. Lopes, C. Moreira, A. Madureira, F. Resende, X. Wu, N. Jayawarna, *et al.*, "Control strategies for microgrids emergency operation," in *2005 International Conference on Future Power Systems*, 2005, pp. 6 pp.-6.

[10] M. C. Chandorkar, D. M. Divan, and R. Adapa, "Control of parallel connected inverters in standalone AC supply systems," *IEEE Transactions on Industry Applications*, vol. 29, pp. 136-143, 1993.

[11] A. Engler, "Applicability of droops in low voltage grids," *International Journal of Distributed Energy Resources*, vol. 1, pp. 1-6, 2005.

[12] J. P. Lopes, C. Moreira, and A. Madureira, "Defining control strategies for microgrids islanded operation," *IEEE Transactions on power systems*, vol. 21, pp. 916-924, 2006.

[13] J. M. Guerrero, J. C. Vasquez, J. Matas, L. G. De Vicuña, and M. Castilla, "Hierarchical control of droop-controlled AC and DC microgrids—A general approach toward standardization," *IEEE Transactions on industrial electronics*, vol. 58, pp. 158-172, 2010.

[14] E. Barklund, N. Pogaku, M. Prodanovic, C. Hernandez-Aramburo, and T. C. Green, "Energy management in autonomous microgrid using stability-constrained droop control of inverters," *IEEE Transactions on Power Electronics*, vol. 23, pp. 2346-2352, 2008.

[15] Y. Li, D. M. Vilathgamuwa, and P. C. Loh, "Design, analysis, and real-time testing of a controller for multibus microgrid system," *IEEE Transactions on power electronics*, vol. 19, pp. 1195-1204, 2004.

[16] R. H. Lasseter, "Microgrids," in *2002 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No. 02CH37309)*, 2002, pp. 305-308.

[17] J. M. Guerrero, J. C. Vasquez, J. Matas, M. Castilla, and L. G. de Vicuña, "Control strategy for flexible microgrid based on parallel line-interactive UPS systems," *IEEE Transactions on industrial Electronics*, vol. 56, pp. 726-736, 2008.
[18] H. Han, X. Hou, J. Yang, J. Wu, M. Su, and J. M. Guerrero, "Review of power sharing control strategies for islanding operation of AC microgrids," *IEEE Transactions on Smart Grid*, vol. 7, pp. 200-215, 2015.
[19] H. Saadat, "Power System Analysis,(2nd)," *McGraw-Hill Higher Education*, 2009.
[20] D. P. Kothari and I. Nagrath, *Modern power system analysis*: Tata McGraw-Hill Education, 1989.
[21] J. A. Mueller and J. W. Kimball, "An Efficient Method of Determining Operating Points of Droop-Controlled Microgrids," *IEEE Transactions on Energy Conversion*, vol. 32, pp. 1432-1446, 2017.
[22] M. M. A. Abdelaziz, H. E. Farag, E. F. El-Saadany, and Y. A.-R. I. Mohamed, "A novel and generalized three-phase power flow algorithm for islanded microgrids using a newton trust region method," *IEEE Transactions on Power Systems*, vol. 28, pp. 190-201, 2013.
[23] W. F. Tinney and C. E. Hart, "Power flow solution by Newton's method," *IEEE Transactions on Power Apparatus and systems*, pp. 1449-1460, 1967.
[24] J. J. Grainger and W. D. Stevenson Jr, "Power System Analisis, Mc GrawHill," *New York*, 1994.
[25] B. Stott, "Decoupled Newton load flow," *IEEE Transactions on Power Apparatus and Systems*, pp. 1955-1959, 1972.
[26] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE transactions on power apparatus and systems*, pp. 859-869, 1974.
[27] M. Haque, "A general load flow method for distribution systems," *Electric Power Systems Research*, vol. 54, pp. 47-54, 2000.
[28] U. Eminoglu and M. H. Hocaoglu, "Distribution systems forward/backward sweep-based power flow algorithms: a review and comparison study," *Electric Power Components and Systems*, vol. 37, pp. 91-110, 2008.
[29] W. Kersting, "Application of Labber Network Theory to the Solution of Three-Phase radial Load Problems," in *IEEE PES winter meeting*, 1976.
[30] R. Berg, E. Hawkins, and W. Pleines, "Mechanized calculation of unbalanced load flow on radial distribution circuits," *IEEE Transactions on power apparatus and systems*, pp. 415-421, 1967.
[31] D. Shirmohammadi, H. W. Hong, A. Semlyen, and G. Luo, "A compensation-based power flow method for weakly meshed distribution and transmission networks," *IEEE Transactions on power systems*, vol. 3, pp. 753-762, 1988.
[32] Y. Zhu and K. Tomsovic, "Adaptive power flow method for distribution systems with dispersed generation," *IEEE Transactions on Power Delivery*, vol. 17, pp. 822-827, 2002.
[33] S. Khushalani, J. M. Solanki, and N. N. Schulz, "Development of three-phase unbalanced power flow using PV and PQ models for distributed generation and study of the impact of DG models," *IEEE Transactions on Power Systems*, vol. 22, pp. 1019-1025, 2007.
[34] K. Balamurugan and D. Srinivasan, "Review of power flow studies on distribution network with distributed generation," in *2011 IEEE Ninth International Conference on Power Electronics and Drive Systems*, 2011, pp. 411-417.

[35] K. P. Wong, A. Li, and M. Law, "Development of constrained-genetic-algorithm load-flow method," *IEE proceedings-Generation, Transmission and Distribution,* vol. 144, pp. 91-99, 1997.
[36] K. P. Wong, A. Li, and T. Law, "Advanced, constrained, genetic algorithm load flow method," *IEE Proceedings-Generation, Transmission and Distribution,* vol. 146, pp. 609-616, 1999.
[37] T. Ting, K. Wong, and C. Chung, "Hybrid constrained genetic algorithm/particle swarm optimisation load flow algorithm," *IET generation, transmission & distribution,* vol. 2, pp. 800-812, 2008.
[38] M. Varadarajan and K. S. Swarup, "Solving multi-objective optimal power flow using differential evolution," *IET Generation, Transmission & Distribution,* vol. 2, pp. 720-730, 2008.
[39] H. Nikkhajoei and R. Iravani, "Steady-state model and power flow analysis of electronically-coupled distributed resource units," *IEEE Transactions on Power Delivery,* vol. 22, pp. 721-728, 2007.
[40] M. Z. Kamh and R. Iravani, "Unbalanced model and power-flow analysis of microgrids and active distribution systems," *IEEE Transactions on Power Delivery,* vol. 25, pp. 2851-2858, 2010.
[41] M. Z. Kamh and R. Iravani, "Steady-state model and power-flow analysis of single-phase electronically coupled distributed energy resources," *IEEE Transactions on Power Delivery,* vol. 27, pp. 131-139, 2011.
[42] M. Z. Kamh and R. Iravani, "A unified three-phase power-flow analysis model for electronically coupled distributed energy resources," *IEEE Transactions on Power Delivery,* vol. 26, pp. 899-909, 2011.
[43] F. Mumtaz, M. Syed, M. Al Hosani, and H. Zeineldin, "A novel approach to solve power flow for islanded microgrids using modified newton raphson with droop control of dg," *IEEE Transactions on Sustainable Energy,* vol. 7, pp. 493-503, 2016.
[44] F. Mumtaz, M. Syed, M. Al Hosani, and H. Zeineldin, "A simple and accurate approach to solve the power flow for balanced islanded microgrids," in *2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC)*, 2015, pp. 1852-1856.
[45] X.-S. Yang, *Nature-inspired optimization algorithms*: Elsevier, 2014.
[46] S. Saremi, S. Mirjalili, and A. Lewis, "Grasshopper optimisation algorithm: theory and application," *Advances in Engineering Software,* vol. 105, pp. 30-47, 2017.
[47] A. Elrayyah, Y. Sozer, and M. E. Elbuluk, "A novel load-flow analysis for stable and optimized microgrid operation," *IEEE Transactions on Power Delivery,* vol. 29, pp. 1709-1717, 2014.
[48] M. Abedini, "A novel algorithm for load flow analysis in island microgrids using an improved evolutionary algorithm," *International Transactions on Electrical Energy Systems,* vol. 26, pp. 2727-2743, 2016.
[49] H. Jiayi, J. Chuanwen, and X. Rong, "A review on distributed energy resources and MicroGrid," *Renewable and Sustainable Energy Reviews,* vol. 12, pp. 2472-2483, 2008.
[50] N. Pogaku, M. Prodanovic, and T. C. Green, "Modeling, analysis and testing of autonomous operation of an inverter-based microgrid," *IEEE Transactions on power electronics,* vol. 22, pp. 613-625, 2007.

[51] M. Rasheduzzaman, J. A. Mueller, and J. W. Kimball, "An accurate small-signal model of inverter-dominated islanded microgrids using $ dq $ reference frame," *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 2, pp. 1070-1080, 2014.
[52] P. C. Krause, O. Wasynczuk, S. D. Sudhoff, and S. Pekarek, *Analysis of electric machinery and drive systems* vol. 2: Wiley Online Library, 2002.
[53] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.
[54] X.-S. Yang, *Nature-inspired metaheuristic algorithms*: Luniver press, 2010.
[55] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan press, 1975.
[56] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*: MIT press, 1992.
[57] A. H. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of genetic algorithms*. vol. 1, ed: Elsevier, 1991, pp. 205-218.
[58] K. Deep and M. Thakur, "A new crossover operator for real coded genetic algorithms," *Applied mathematics and computation*, vol. 188, pp. 895-911, 2007.
[59] Y. Yoon and Y.-H. Kim, "The roles of crossover and mutation in real-coded genetic algorithms," in *Bio-inspired computational algorithms and their applications*, ed: IntechOpen, 2012.
[60] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, pp. 341-359, 1997.
[61] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE transactions on evolutionary computation*, vol. 15, pp. 4-31, 2010.
[62] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, 1995, pp. 39-43.
[63] J. Kennedy, "Particle Swarm Optimization," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., ed Boston, MA: Springer US, 2010, pp. 760-766.
[64] M. S. Arumugam, M. Rao, and A. Chandramohan, "A new and improved version of particle swarm optimization algorithm with global–local best parameters," *Knowledge and Information systems*, vol. 16, pp. 331-357, 2008.
[65] S. Y. Lim, M. Montakhab, and H. Nouri, "Economic dispatch of power system using particle swarm optimization with constriction factor," *International Journal of Innovations in Energy Systems and Power*, vol. 4, pp. 29-34, 2009.
[66] Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Proceedings of the 2001 congress on evolutionary computation (IEEE Cat. No. 01TH8546)*, 2001, pp. 81-86.
[67] E. Atashpaz-Gargari and C. Lucas, "Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition," in *2007 IEEE congress on evolutionary computation*, 2007, pp. 4661-4667.
[68] S. Hosseini and A. Al Khaled, "A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research," *Applied Soft Computing*, vol. 24, pp. 1078-1094, 2014.
[69] X.-S. Yang, "Flower pollination algorithm for global optimization," in *International conference on unconventional computing and natural computation*, 2012, pp. 240-249.

[70] E. Nabil, "A modified flower pollination algorithm for global optimization," *Expert Systems with Applications,* vol. 57, pp. 192-203, 2016.
[71] I. Pavlyukevich, "Lévy flights, non-local search and simulated annealing," *Journal of Computational Physics,* vol. 226, pp. 1830-1844, 2007.
[72] S. M. Ismael, S. H. A. Aleem, A. Y. Abdelaziz, and A. F. Zobaa, "Optimal Conductor Selection of Radial Distribution Feeders: An Overview and New Application Using Grasshopper Optimization Algorithm," in *Classical and Recent Aspects of Power System Optimization*, ed: Elsevier, 2018, pp. 185-217.
[73] L. Luo and S. V. Dhople, "Spatiotemporal model reduction of inverter-based islanded microgrids," *IEEE Transactions on Energy Conversion,* vol. 29, pp. 823-832, 2014.
[74] IEEE, "IEEE Guide for Identifying and Improving Voltage Quality in Power Systems - Redline," ed. IEEE Std 1250-2011 (Revision of IEEE Std 1250-1995): IEEE, 2011, pp. 1-70.
[75] Available: https://yarpiz.com/

Appendix

MATLAB Code of ICDE

```
% Main Body of ICDE
clc;
clear;
close all;

for re = 1:30       % for 30 independent trials

tic

%% Problem Definition

CostFunction = @(x) MG_TEST_37(x);        % Cost Function

nVar=14;             % Number of Decision Variables

VarSize=[1 nVar];   % Decision Variables Matrix Size

wmin = 375;
wmax = 379;
d_min = -3; d_max = 3; v_min = 160; v_max = 180;

VarMin = [wmin d_min d_min d_min d_min d_min d_min v_min v_min v_min v_min
v_min v_min v_min];         % Lower Bound of Variables
VarMax = [wmax d_max d_max d_max d_max d_max d_max v_max v_max v_max v_max
v_max v_max v_max];         % Upper Bound of Variables

%% ICA Parameters

MaxIt=100;          % Maximum Number of Iterations

nPop=100;           % Population Size
nEmp=5;             % Number of Empires/Imperialists

alpha=1;            % Selection Pressure

beta=1.5;           % Assimilation Coefficient

pRevolution=0.05;   % Revolution Probability
mu=0.1;             % Revolution Rate

zeta=0.2;           % Colonies Mean Cost Coefficient

%% Globalization of Parameters and Settings

global ProblemSettings;
ProblemSettings.CostFunction=CostFunction;
ProblemSettings.nVar=nVar;
ProblemSettings.VarSize=VarSize;
ProblemSettings.VarMin=VarMin;
ProblemSettings.VarMax=VarMax;
```

```
global ICASettings;
ICASettings.MaxIt=MaxIt;
ICASettings.nPop=nPop;
ICASettings.nEmp=nEmp;
ICASettings.alpha=alpha;
ICASettings.beta=beta;
ICASettings.pRevolution=pRevolution;
ICASettings.mu=mu;
ICASettings.zeta=zeta;

%% Initialization

% Initialize Empires
[emp, country]=CreateInitialEmpires();

% Array to Hold Best Cost Values
BestCost=zeros(MaxIt,1);

%% ICA Main Loop

for it=1:MaxIt

    % Assimilation
    emp=AssimilateColonies(emp);

    % Revolution
    emp=DoRevolution(emp);

    % Intra-Empire Competition
    emp=IntraEmpireCompetition(emp);

    % Differential Evolution
    emp=Differential_Evolution(emp);

    % Intra-Empire Competition
    emp=IntraEmpireCompetition(emp);

    % Update Total Cost of Empires
    emp=UpdateTotalCost(emp);

    % Inter-Empire Competition
    emp=InterEmpireCompetition(emp);

    % Update Best Solution Ever Found
    imp=[emp.Imp];
    [~, BestImpIndex]=min([imp.Cost]);
    BestSol=imp(BestImpIndex);

    % Update Best Cost
    BestCost(it)=BestSol.Cost;

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

    if BestCost(it) < 1e-5
        break;
```

```
    end
end

%% Results

duration = toc

xlswrite('ICDE_Load_Flow_Results.xlsx',[BestSol.Position,BestSol.Cost,it,dura
tion],['A' num2str(re) ':' 'Q' num2str(re)])
xlswrite('ICDE_fitness_per_trial.xlsx',[it,BestCost'],['A' num2str(re) ':'
'CW' num2str(re)])
re
end
```

```
                        % Code for objective function

function z=MG_TEST_37(x)

wn = 2*pi*60;
Vn = 170;
m1 = (1/2387.5); m2 = (1/1273.2); m3 = (1/1591.5); m4 = (1/2387.5); ...
m5 = (1/1273.2); m6 = (1/1591.5); m7 = (1/2387.5);
n1 = (1/1250); n2 = (1/666.7); n3 = (1/833.3); n4 = (1/1250); ...
n5 = (1/666.7); n6 = (1/833.3); n7 = (1/1250);

R1_2  = 0.167; R2_5  = 0.070; R2_13 = 0.063; R2_3  = 0.230; R3_20 = 0.042;
R3_23 = 0.105; R4_14 = 0.014; R4_16 = 0.139; R5_34 = 0.056; R5_12 = 0.042;
R6_19 = 0.049; R7_18 = 0.132; R7_17 = 0.021; R8_26 = 0.056; R8_25 = 0.056;
R9_24 = 0.105; R9_8  = 0.056; R10_28= 0.035; R10_29= 0.233; R11_33= 0.070;
R11_32= 0.035; R13_4 = 0.091; R14_15= 0.091; R16_7 = 0.160; R16_6 = 0.105;
R20_35= 0.049; R23_9 = 0.035; R26_27= 0.098; R27_30= 0.112; R27_10= 0.091;
R30_31= 0.070; R31_11= 0.070; R35_21= 0.035; R35_22= 0.049; R36_9 = 0.230;

L1_2  = 2.31e-4; L2_5  = 9.64e-5; L2_13 = 8.67e-5; L2_3  = 3.18e-4; L3_20 =
5.78e-5;
L3_23 = 1.45e-4; L4_14 = 1.93e-5; L4_16 = 1.93e-4; L5_34 = 7.71e-5; L5_12 =
5.78e-5;
L6_19 = 6.75e-5; L7_18 = 1.83e-4; L7_17 = 2.89e-5; L8_26 = 7.71e-5; L8_25 =
7.71e-5;
L9_24 = 1.45e-4; L9_8  = 7.71e-5; L10_28= 4.82e-5; L10_29= 3.08e-4; L11_33=
9.64e-5;
L11_32= 4.82e-5; L13_4 = 1.25e-4; L14_15= 1.25e-4; L16_7 = 2.22e-4; L16_6 =
1.45e-4;
L20_35= 6.75e-5; L23_9 = 4.82e-5; L26_27= 1.35e-4; L27_30= 1.54e-4; L27_10=
1.25e-4;
L30_31= 9.64e-5; L31_11= 9.64e-5; L35_21= 4.82e-5; L35_22= 6.75e-5; L36_9 =
3.18e-4;

Rload1  =  6.58; Rload12 = 49.93; Rload13 = 49.93; Rload14 = 111.41; Rload15
= 49.93;
Rload16 = 49.93; Rload17 = 25.82; Rload18 = 98.74; Rload19 =  98.74; Rload20
= 98.74;
Rload21 = 32.91; Rload22 = 98.74; Rload23 = 49.93; Rload24 =  49.93; Rload25
= 98.74;
Rload26 = 49.93; Rload27 = 98.74; Rload28 = 49.93; Rload29 =  98.74; Rload30
= 29.62;
```

```
Rload31 = 33.12; Rload32 = 49.93; Rload33 = 98.74; Rload34 =  45.54; Rload35
= 98.74;

Lload1  = 0.0105; Lload12 = 0.0748; Lload13 = 0.0748; Lload14 = 0.1681;
Lload15 = 0.0748;
Lload16 = 0.0748; Lload17 = 0.0409; Lload18 = 0.1572; Lload19 = 0.1572;
Lload20 = 0.1572;
Lload21 = 0.0524; Lload22 = 0.1572; Lload23 = 0.0748; Lload24 = 0.0748;
Lload25 = 0.1572;
Lload26 = 0.0748; Lload27 = 0.1572; Lload28 = 0.0748; Lload29 = 0.1572;
Lload30 = 0.0472;
Lload31 = 0.0519; Lload32 = 0.0748; Lload33 = 0.1572; Lload34 = 0.0686;
Lload35 = 0.1572;

Rc = 0.1; Lc = 0.5e-3;

imp_line = [0 1 Rload1 Lload1; 0 12 Rload12 Lload12; 0 13 Rload13 Lload13; 0
14 Rload14 Lload14;
            0 15 Rload15 Lload15; 0 16 Rload16 Lload16; 0 17 Rload17 Lload17;
0 18 Rload18 Lload18;
            0 19 Rload19 Lload19; 0 20 Rload20 Lload20; 0 21 Rload21 Lload21;
0 22 Rload22 Lload22;
            0 23 Rload23 Lload23; 0 24 Rload24 Lload24; 0 25 Rload25 Lload25;
0 26 Rload26 Lload26;
            0 27 Rload27 Lload27; 0 28 Rload28 Lload28; 0 29 Rload29 Lload29;
0 30 Rload30 Lload30;
            0 31 Rload31 Lload31; 0 32 Rload32 Lload32; 0 33 Rload33 Lload33;
0 34 Rload34 Lload34;
            0 35 Rload35 Lload35;
            0 15 Rc Lc; 0 18 Rc Lc; 0 22 Rc Lc; 0 24 Rc Lc; 0 29 Rc Lc; 0 33
Rc Lc; 0 34 Rc Lc;
            1 2 R1_2 L1_2; 2 5 R2_5 L2_5; 2 13 R2_13 L2_13; 2 3 R2_3 L2_3; 3
20 R3_20 L3_20;
            3 23 R3_23 L3_23; 4 14 R4_14 L4_14; 4 16 R4_16 L4_16; 5 34 R5_34
L5_34; 5 12 R5_12 L5_12;
            6 19 R6_19 L6_19; 7 18 R7_18 L7_18; 7 17 R7_17 L7_17; 8 26 R8_26
L8_26; 8 25 R8_25 L8_25;
            9 24 R9_24 L9_24; 9 8 R9_8 L9_8; 10 28 R10_28 L10_28; 10 29
R10_29 L10_29; 11 33 R11_33 L11_33;
            11 32 R11_32 L11_32; 13 4 R13_4 L13_4; 14 15 R14_15 L14_15; 16 7
R16_7 L16_7; 16 6 R16_6 L16_6;
            20 35 R20_35 L20_35; 23 9 R23_9 L23_9; 26 27 R26_27 L26_27; 27 30
R27_30 L27_30; 27 10 R27_10 L27_10;
            30 31 R30_31 L30_31; 31 11 R31_11 L31_11; 35 21 R35_21 L35_21; 35
22 R35_22 L35_22; 36 9 R36_9 L36_9];

del1 = 0;

               y = x;

               [Zbus] = zbuild_w(imp_line,y(1));

               Zc = Rc + j*y(1)*Lc;  %coupling impedance

               %Complex inverters voltages
               V1d = (y(8))*sin(del1);
               V1q = (y(8))*cos(del1);
```

```
                V2d = (y(9))*sin((y(2)*pi)/180);
                V2q = (y(9))*cos((y(2)*pi)/180);

                V3d = (y(10))*sin((y(3)*pi)/180);
                V3q = (y(10))*cos((y(3)*pi)/180);

                V4d = (y(11))*sin((y(4)*pi)/180);
                V4q = (y(11))*cos((y(4)*pi)/180);

                V5d = (y(12))*sin((y(5)*pi)/180);
                V5q = (y(12))*cos((y(5)*pi)/180);

                V6d = (y(13))*sin((y(6)*pi)/180);
                V6q = (y(13))*cos((y(6)*pi)/180);

                V7d = (y(14))*sin((y(7)*pi)/180);
                V7q = (y(14))*cos((y(7)*pi)/180);

                V1 = V1d + j*V1q;
                V2 = V2d + j*V2q;
                V3 = V3d + j*V3q;
                V4 = V4d + j*V4q;
                V5 = V5d + j*V5q;
                V6 = V6d + j*V6q;
                V7 = V7d + j*V7q;

Iinj = zeros(36,1);

Isc1 = V1/Zc;
Isc2 = V2/Zc;
Isc3 = V3/Zc;
Isc4 = V4/Zc;
Isc5 = V5/Zc;
Isc6 = V6/Zc;
Isc7 = V7/Zc;

Iinj(15) = Isc1;
Iinj(18) = Isc2;
Iinj(22) = Isc3;
Iinj(24) = Isc4;
Iinj(29) = Isc5;
Iinj(33) = Isc6;
Iinj(34) = Isc7;

Vb = Zbus*Iinj;     %since Iex = 0, Isc = Iinj

Vb1 = Vb(15);
Vb2 = Vb(18);
Vb3 = Vb(22);
Vb4 = Vb(24);
Vb5 = Vb(29);
Vb6 = Vb(33);
Vb7 = Vb(34);

                %inverters currents
I01 = (V1 - Vb1)/Zc;
```

```
I02 = (V2 - Vb2)/Zc;
I03 = (V3 - Vb3)/Zc;
I04 = (V4 - Vb4)/Zc;
I05 = (V5 - Vb5)/Zc;
I06 = (V6 - Vb6)/Zc;
I07 = (V7 - Vb7)/Zc;

I01d = real(I01);
I01q = imag(I01);

I02d = real(I02);
I02q = imag(I02);

I03d = real(I03);
I03q = imag(I03);

I04d = real(I04);
I04q = imag(I04);

I05d = real(I05);
I05q = imag(I05);

I06d = real(I06);
I06q = imag(I06);

I07d = real(I07);
I07q = imag(I07);

%inverters power mismatch

dP1 = 3/2*(V1d*I01d + V1q*I01q) - (wn - y(1))/m1;
dP2 = 3/2*(V2d*I02d + V2q*I02q) - (wn - y(1))/m2;
dP3 = 3/2*(V3d*I03d + V3q*I03q) - (wn - y(1))/m3;
dP4 = 3/2*(V4d*I04d + V4q*I04q) - (wn - y(1))/m4;
dP5 = 3/2*(V5d*I05d + V5q*I05q) - (wn - y(1))/m5;
dP6 = 3/2*(V6d*I06d + V6q*I06q) - (wn - y(1))/m6;
dP7 = 3/2*(V7d*I07d + V7q*I07q) - (wn - y(1))/m7;
dQ1 = 3/2*(V1q*I01d - V1d*I01q) - (Vn - y(8))/n1;
dQ2 = 3/2*(V2q*I02d - V2d*I02q) - (Vn - y(9))/n2;
dQ3 = 3/2*(V3q*I03d - V3d*I03q) - (Vn - y(10))/n3;
dQ4 = 3/2*(V4q*I04d - V4d*I04q) - (Vn - y(11))/n4;
dQ5 = 3/2*(V5q*I05d - V5d*I05q) - (Vn - y(12))/n5;
dQ6 = 3/2*(V6q*I06d - V6d*I06q) - (Vn - y(13))/n6;
dQ7 = 3/2*(V7q*I07d - V7d*I07q) - (Vn - y(14))/n7;

z = abs(dP1 + dP2 + dP3 + dP4 + dP5 + dP6 + dP7) + abs(dQ1 + dQ2 + dQ3 + dQ4
+ dQ5 + dQ6 + dQ7);
end
```

% Code for formation of Zbus matrix

```
function [Zbus] = zbuild_w(linedata,w)
nl = linedata(:,1); nr = linedata(:,2); R = linedata(:,3);
L = linedata(:,4);
X = w*L;
nbr=length(linedata(:,1)); nbus = max(max(nl), max(nr));
for k=1:nbr
```

```
    if R(k) == inf | X(k) ==inf
    R(k) = 99999999; X(k) = 99999999;
    else, end
end

ZB = R + j*X;
Zbus = zeros(nbus, nbus);
tree=0;  %%%%new
% Adding a branch from a new bus to reference bus 0
  for I = 1:nbr
  ntree(I) = 1;
      if nl(I) == 0 | nr(I) == 0
          if nl(I) == 0      n = nr(I);
          elseif nr(I) == 0  n = nl(I);
          end
              if abs(Zbus(n, n)) == 0   Zbus(n,n) = ZB(I);tree=tree+1; %%new
              else Zbus(n,n) = Zbus(n,n)*ZB(I)/(Zbus(n,n) + ZB(I));
              end
              ntree(I) = 2;
      else,end
  end

% Adding a branch from new bus to an existing bus
while tree < nbus  %%% new

  for n = 1:nbus
     nadd = 1;
     if abs(Zbus(n,n)) == 0
        for I = 1:nbr
            if nadd == 1;
                if nl(I) == n | nr(I) == n
                   if nl(I) == n      k = nr(I);
                   elseif nr(I) == n  k = nl(I);
                   end
                     if abs(Zbus(k,k)) ~= 0
                          for m = 1:nbus
                              if m ~= n
                              Zbus(m,n) = Zbus(m,k);
                              Zbus(n,m) = Zbus(m,k);
                              else, end
                          end
                          Zbus(n,n) = Zbus(k,k) + ZB(I); tree=tree+1; %%new
                          nadd = 2;  ntree(I) = 2;
                     else, end
                else, end
            else, end
        end
     else, end
end


end  %%%%%%new

% Adding a link between two old buses
  for n = 1:nbus
      for I = 1:nbr
          if ntree(I) == 1
```

```
            if nl(I) == n | nr(I) == n
                if nl(I) == n     k = nr(I);
                elseif nr(I) == n k = nl(I);
                end
            DM = Zbus(n,n) + Zbus(k,k) + ZB(I) - 2*Zbus(n,k);
                 for jj = 1:nbus
                 AP = Zbus(jj,n) - Zbus(jj,k);
                      for kk = 1:nbus
                      AT = Zbus(n,kk) - Zbus(k, kk);
                      DELZ(jj,kk) = AP*AT/DM;
                      end
                 end
                 Zbus = Zbus - DELZ;
                 ntree(I) = 2;
            else,end
         else,end
      end
  end
```

```
                  % Function for initializing the empires

function [emp, country]=CreateInitialEmpires()

    global ProblemSettings;
    global ICASettings;

    CostFunction=ProblemSettings.CostFunction;
    nVar=ProblemSettings.nVar;
    VarSize=ProblemSettings.VarSize;
    VarMin=ProblemSettings.VarMin;
    VarMax=ProblemSettings.VarMax;

    nPop=ICASettings.nPop;
    nEmp=ICASettings.nEmp;
    nCol=nPop-nEmp;
    alpha=ICASettings.alpha;

    empty_country.Position=[];
    empty_country.Cost=[];

    country=repmat(empty_country,nPop,1);

    for i=1:nPop
        for j = 1:nVar
        country(i).Position(j)=unifrnd(VarMin(j),VarMax(j));
        end

        country(i).Cost=CostFunction(country(i).Position);

    end

    costs=[country.Cost];
    [~, SortOrder]=sort(costs);

    country=country(SortOrder);

    imp=country(1:nEmp);
```

```
    col=country(nEmp+1:end);


    empty_empire.Imp=[];
    empty_empire.Col=repmat(empty_country,0,1);
    empty_empire.nCol=0;
    empty_empire.TotalCost=[];

    emp=repmat(empty_empire,nEmp,1);

    % Assign Imperialists
    for k=1:nEmp
        emp(k).Imp=imp(k);
    end

    % Assign Colonies
    P=exp(-alpha*[imp.Cost]/max([imp.Cost]));
    P=P/sum(P);
    for j=1:nCol

        k=RouletteWheelSelection(P);

        emp(k).Col=[emp(k).Col col(j)];


        emp(k).nCol=emp(k).nCol+1;
    end

    emp=UpdateTotalCost(emp);
end

                                % Assimilation

function emp=AssimilateColonies(emp)

    global ProblemSettings;
    CostFunction=ProblemSettings.CostFunction;
    nVar=ProblemSettings.nVar;
    VarSize=ProblemSettings.VarSize;
    VarMin=ProblemSettings.VarMin;
    VarMax=ProblemSettings.VarMax;

    global ICASettings;
    beta=ICASettings.beta;

    nEmp=numel(emp);
    for k=1:nEmp
        for i=1:emp(k).nCol

            emp(k).Col(i).Position = emp(k).Col(i).Position ...
                + beta*rand(VarSize).*(emp(k).Imp.Position-
emp(k).Col(i).Position);

            for j = 1:nVar
                emp(k).Col(i).Position(j) =
max(emp(k).Col(i).Position(j),VarMin(j));
```

```
                emp(k).Col(i).Position(j) =
min(emp(k).Col(i).Position(j),VarMax(j));
            end

            emp(k).Col(i).Cost = CostFunction(emp(k).Col(i).Position);

        end
    end

end
```

```
                                % Revolution

function emp=DoRevolution(emp)

    global ProblemSettings;
    CostFunction=ProblemSettings.CostFunction;
    nVar=ProblemSettings.nVar;
    VarSize=ProblemSettings.VarSize;
    VarMin=ProblemSettings.VarMin;
    VarMax=ProblemSettings.VarMax;

    global ICASettings;
    pRevolution=ICASettings.pRevolution;
    mu=ICASettings.mu;

    nmu=ceil(mu*nVar);

    sigma=0.1*(VarMax-VarMin);

    nEmp=numel(emp);
    for k=1:nEmp

        NewPos = emp(k).Imp.Position + sigma.*randn(VarSize);

        jj=randsample(nVar,nmu)';
        NewImp=emp(k).Imp;
        NewImp.Position(jj)=NewPos(jj);

        NewImp.Cost=CostFunction(NewImp.Position);
        if NewImp.Cost<emp(k).Imp.Cost
            emp(k).Imp = NewImp;
        end

        for i=1:emp(k).nCol
            if rand<=pRevolution

                NewPos = emp(k).Col(i).Position + sigma.*randn(VarSize);

                jj=randsample(nVar,nmu)';
                emp(k).Col(i).Position(jj) = NewPos(jj);

              for j = 1:nVar
                emp(k).Col(i).Position(j) =
max(emp(k).Col(i).Position(j),VarMin(j));
                emp(k).Col(i).Position(j) =
min(emp(k).Col(i).Position(j),VarMax(j));
```

```
            end
              emp(k).Col(i).Cost = CostFunction(emp(k).Col(i).Position);

        end
    end
end
```

```
                        % Intra-Empire Competition

function emp=IntraEmpireCompetition(emp)

    nEmp=numel(emp);

    for k=1:nEmp
        for i=1:emp(k).nCol
            if emp(k).Col(i).Cost<emp(k).Imp.Cost
                imp=emp(k).Imp;
                col=emp(k).Col(i);

                emp(k).Imp=col;
                emp(k).Col(i)=imp;
            end
        end
    end
end
```

```
                      % Update total cost of empires

function emp=UpdateTotalCost(emp)

    global ICASettings;
    zeta=ICASettings.zeta;

    nEmp=numel(emp);

    for k=1:nEmp
        if emp(k).nCol>0
            emp(k).TotalCost=emp(k).Imp.Cost+zeta*mean([emp(k).Col.Cost]);
        else
            emp(k).TotalCost=emp(k).Imp.Cost;
        end
    end
end
```

```
                        % Imperialistic Competition

function emp=InterEmpireCompetition(emp)

    if numel(emp)==1
        return;
    end

    global ICASettings;
    alpha=ICASettings.alpha;

    TotalCost=[emp.TotalCost];
```

```
    [~, WeakestEmpIndex]=max(TotalCost);
    WeakestEmp=emp(WeakestEmpIndex);

    P=exp(-alpha*TotalCost/max(TotalCost));
    P(WeakestEmpIndex)=0;
    P=P/sum(P);
    if any(isnan(P))
        P(isnan(P))=0;
        if all(P==0)
            P(:)=1;
        end
        P=P/sum(P);
    end

    if WeakestEmp.nCol>0
        [~, WeakestColIndex]=max([WeakestEmp.Col.Cost]);
        WeakestCol=WeakestEmp.Col(WeakestColIndex);

        WinnerEmpIndex=RouletteWheelSelection(P);
        WinnerEmp=emp(WinnerEmpIndex);

        WinnerEmp.Col(end+1)=WeakestCol;
        WinnerEmp.nCol=WinnerEmp.nCol+1;
        emp(WinnerEmpIndex)=WinnerEmp;

        WeakestEmp.Col(WeakestColIndex)=[];
        WeakestEmp.nCol=WeakestEmp.nCol-1;
        emp(WeakestEmpIndex)=WeakestEmp;
    end

    if WeakestEmp.nCol==0

        WinnerEmpIndex2=RouletteWheelSelection(P);
        WinnerEmp2=emp(WinnerEmpIndex2);

        WinnerEmp2.Col(end+1)=WeakestEmp.Imp;
        WinnerEmp2.nCol=WinnerEmp2.nCol+1;
        emp(WinnerEmpIndex2)=WinnerEmp2;

        emp(WeakestEmpIndex)=[];
    end

end

                        % Differential Evolution

function emp = Differential_Evolution(emp)

    global ProblemSettings;
    CostFunction=ProblemSettings.CostFunction;
    nVar=ProblemSettings.nVar;
    VarSize=ProblemSettings.VarSize;
    VarMin=ProblemSettings.VarMin;
    VarMax=ProblemSettings.VarMax;

    beta_min=0.4;   % Lower Bound of Scaling Factor
```

```
    beta_max=1.0;    % Upper Bound of Scaling Factor

    pCR=0.5;         % Crossover Probability

    nEmp=numel(emp);
    for k=1:nEmp

        x = emp(k).Imp.Position;

        A = randperm(nEmp);

        A(A==k) = [];

        a=A(1);
        b=A(2);
        c=A(3);

        % Mutation
        beta = unifrnd(beta_min,beta_max,VarSize);
        y = emp(a).Imp.Position + beta.*(emp(b).Imp.Position -
emp(c).Imp.Position);

        for p = 1:nVar
            y(p) = max(y(p), VarMin(p));
            y(p) = min(y(p), VarMax(p));
        end

        % Crossover
        z=zeros(size(x));
        j0=randi([1 numel(x)]);

        for j=1:numel(x)
            if j==j0 || rand<=pCR
                z(j)=y(j);
            else
                z(j)=x(j);
            end
        end

        NewSol.Position = z;
        NewSol.Cost = CostFunction(NewSol.Position);

        if NewSol.Cost < emp(k).Imp.Cost
            emp(k).Imp = NewSol;
        end

        for i=1:emp(k).nCol
            if emp(k).nCol < 4
                break;
            end
            x = emp(k).Col(i).Position;

            A = randperm(emp(k).nCol);

            A(A==i) = [];

            a=A(1);
```

```
            b=A(2);
            c=A(3);

            % Mutation
            beta = unifrnd(beta_min,beta_max,VarSize);
            y = emp(k).Col(a).Position + beta.*(emp(k).Col(b).Position -
emp(k).Col(c).Position);

            for p = 1:nVar
                y(p) = max(y(p), VarMin(p));
                y(p) = min(y(p), VarMax(p));
            end
            % Crossover
            z=zeros(size(x));
            j0=randi([1 numel(x)]);

            for j=1:numel(x)
                if j==j0 || rand<=pCR
                    z(j)=y(j);
                else
                    z(j)=x(j);
                end
            end
            NewSol.Position = z;
            NewSol.Cost = CostFunction(NewSol.Position);

            if NewSol.Cost < emp(k).Col(i).Cost
                emp(k).Col(i) = NewSol;
            end
        end
    end
end
```