# Hybridization of Differential Evolution with Bacterial Foraging Optimization Technique for enhance Optimization

By

**Saumyadeep Das**

**Registration No- 137291** of **2016-17**

**Examination Roll No – M6IAR19001**

**Under The Guidance of**

**Dr. Pratyusha Rakshit**

*This thesis is submitted in the partial fulfillment for the Degree of Master of Technology in Intelligent Automation & Robotics under Electronics and Telecommunication Engineering.*

*DEPARTMENT OF ELECTRONICS AND TELECOMMUNICATION ENGINEERING*

Jadavpur University

Kolkata – 700032

May 2019

## FACULTY OF ENGINEERING AND TECHNOLOGY

## JADAVPUR UNIVERSITY

### CERTIFICATE OF RECOMMANDATION

This is to certify that the dissertation entitled, "**Hybridization of Differential Evolution With Bacterial Foraging Optimization Technique for Enhanced Optimization**" has been carried out by SAUMYADEEP DAS (University Registration No.: 137291 of 2016-2017 ) under my guidance and supervision and be accepted in partial fulfillment of the requirement for the Degree of Master of Electronics & Telecommunication Engineering. The research results presented in the thesis have not been included in any other paper submitted for the award of any degree to any other University or Institute.

Dr. Pratyusha Rakshit
(Project Guide)
Dept. of Electronics & Telecommunication
Engineering, Jadavpur University

Prof. Amit Konar
(Course Co-ordinator)
Intelligent Automation & Robotics
Dept. of Electronics & Telecommunication
Engineering, Jadavpur University

Prof. Sheli Sinha Chaudhuri
(Head of the Department)
Dept. of Electronics & Telecommunication
Engineering, Jadavpur University

Prof. Chiranjib Bhattacharjee
Dean, Faculty Council of Engineering
and Technology, Jadavpur University

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**JADAVPUR UNIVERSITY**

---

**CERTIFICATE OF APPROVAL**

The forgoing thesis is hereby approved as a creditable study of an engineering subject and presented in a manner satisfactory to warrant acceptance as prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for which it is submitted.

**Saumyadeep Das**

Examination Roll Number: **M6IAR19001**

**Committee on final examination for evaluation of the thesis**

External Examiner

Dr. Pratyusha Rakshit

**FACULTY OF ENGINEERING AND TECHNOLOGY**

**JADAVPUR UNIVERSITY**

---

<u>DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC THESIS</u>

I hereby declare that this thesis entitled, **"Hybridization of Differential Evolution With Bacterial Foraging Optimization Technique for Enhanced Optimization**", contains literature survey & original research work by the undersigned candidate, as part of his Degree of Master of Technology in *Intelligent Automation & Robotics.*

All information have been obtained and presented in accordance with academic rules & ethical conduct.

I also declare that, as required by these rules & conduct, I have fully cited & reference all materials when required and none of the work represented in this thesis is fabricated.

**Name**: Saumyadeep Das

**Examination Roll No**: M6IAR19001

**Thesis Title**: Hybridization of Differential Evolution With Bacterial Foraging Optimization Technique for Enhanced Optimization

Date:

Place: Kolkata

Signature of the candidate

# PREFACE

The main purpose of any kind of optimization is to achieve best design with respect to a set of prioritized criteria or constraints. In engineering aspect, these include maximizing factors such as productivity, strength, reliability, longevity, efficiency, and utilization.

High-speed digital computers made implementation of the complex optimization procedures possible and stimulated further research on newer methods. Simulated annealing, evolutionary algorithms including genetic algorithms, and neural network methods represent a new class of mathematical programming techniques that have come into prominence during the last decade.

Two powerful optimization techniques are Differential Evolution & Bacterial Foraging Optimization. But they are alone not sufficient to optimize a particular system design due to their limitation in performance. In order to prevent stagnation, and more generally, to reach a high standard of performance in a DE framework, several studies have been carried out in recent years. Numerical results show that a DE can perform very well but requires a very extensive a priori study in order to tune the parameters and in particular the population size. A proper hybridization of various algorithmic components (MA) leads to an optimization algorithm which performs better than its components separately and finely tuned.

# TABLE OF CONTENTS

## LIST OF FIGURES

**Page No.**

# LIST OF TABLES

**Page No.**

# *CHAPTER 1*

## INTRODUCTION

*This chapter gives a brief overview of optimization technique, traditional optimization techniques, and general overview of two important optimization tools i.e., Differential Evolution & Swarm Optimization which are advantageous over other optimization techniques. The requirement of hybridization of two optimization techniques & why it is advantageous are also discussed briefly in this chapter.*

## 1.1 Optimization-

Optimization is an important tool for selection of best element with regard to some conditions & criteria which are used everywhere, from Railway seat reservation to Banking, finance & from engineering design to internet routing. In reality when we try to maximize our profit by keeping the cost minimum, when we try to design a system which will consume less energy by giving the best in class efficiency & output then the term 'Optimization' plays an important role. Basically, we have a limited time, money & resources and this is the reason why optimization is indispensable.

An optimization algorithm computes best available information & values of an objective function given different type of values of different domains. The fulcrum of an optimization process is different search algorithms & computational modeling. Mathematical modeling & Search algorithms are key tools of achieving optimality of the problems of interest.

## 1.2 Traditional Optimization Techniques –

There are several traditional optimization techniques are presents. According to S Koziel[1], algorithms for optimization are more diverse than the types of optimization, though
the right choice of algorithms is an important issue. These optimization algorithms can be classified into classes as below:-

a) Derivative based algorithm
b) Derivative free algorithm
c) Meta-heuristic algorithm

- Derivative based algorithms can be further classified as-
    1. Newton's method & hill climbing
    2. Conjugate gradient method
- Derivative free algorithms can be further classified as-
    1. Pattern search
    2. Trust region method
- Meta heuristic algorithms can be classified as-
    1. Simulated Annealing
    2. Genetic algorithm & differential evolution

3. Particle swarm optimization
4. Harmony search
5. Firefly algorithm
6. Cuckoo search

## 1.3 Evolutionary & swarm optimization techniques-

In this section two algorithms among the different type of optimization algorithms are discussed as these algorithms are very popular & they also have some advantages over traditional optimization techniques.

According to Dr. D.B.Fogel[2] ,a pioneer in evolutionary computation, the problems like robotics, operation research, decision making, bioinformatics, machine learning, data mining and many more are very complex and hard to solve. An approach to face up such real life complex problems inspired by Darwinian natural evolution is referred as Evolutionary Computations. Evolutionary computing involves various algorithms; commonly known as Evolutionary Algorithms. During last two decades Evolutionary Algorithms becomes very popular tool for searching, optimization and providing solutions to complex problems. Evolutionary technique is best suited to the applications where it is not possible to use heuristic solutions and may lead to inadequate results.

1.3.1 Evolutionary optimization advantages-

### a. Conceptual Simplicity-

The prime advantage of an evolutionary optimization technique is the simplicity in concept. A flow chart is shown which describes the evolutionary function applied for function-

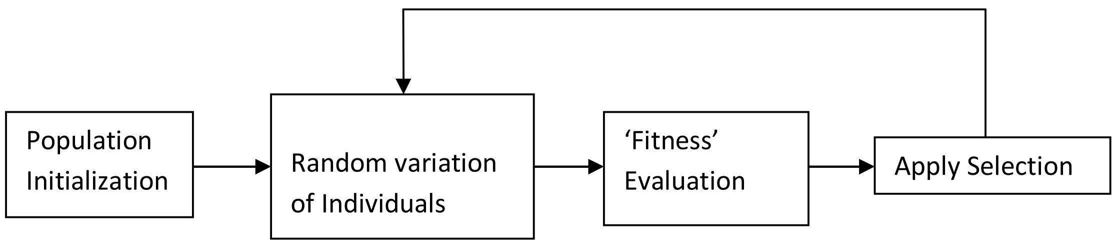


*Figure: 1.1 Differential Evolution Basic Block Diagram*

A population of candidate solutions to a problem at hand is initialized. This is done by randomly sampling from the space of possible solutions. New solutions are created by randomly varying existing solutions. This random variation may include mutation and/or recombination. Competing solutions are evaluated in the basis of a performance index describing their "fitness". Selection is then applied to determine which solutions will be maintained into the next generation, and with what frequency. These new "parents" are then subjected to random variation, and the process iterates.

### b. Broad Applicability-

Evolutionary techniques can be applied to virtually any problem that can be formulated as a function optimization task. It requires a data structure to represent solutions, a performance index to evaluate solutions, and variation operators to generate new solutions from old solutions

### c. Outperform Classic Methods on Real Problems-

optimization problems often (1) impose nonlinear constraints, (2) require payoff functions that are not concerned with least squared error, (3) involve non stationary conditions, (4) incorporate noisy observations or random processing, or include other vagaries that do not conform well to the prerequisites of classic optimization techniques. In addition, in the often encountered case of applying linear programming to problems with nonlinear constraints, this offers an almost certainly incorrect result because the assumptions required for the technique are violated. In contrast, evolutionary computation can directly incorporate arbitrary constraints.

### d. Potential to Use Knowledge and Hybridize with other Methods-

It is always reasonable to incorporate domain-specific knowledge into an algorithm when addressing particular real-world problems. Specialized algorithms can outperform unspecialized algorithms on a restricted domain of interest (Wolpert and Macready, 1997). Evolutionary algorithms offer a framework such that it is comparably easy to incorporate such knowledge. For example, specific variation operators may be known to be useful when applied to particular representations.

These can be directly applied as mutation or recombination operations. Knowledge can also be implemented into the performance index, in the form of known physical or chemical properties (e.g., van der Waals interactions, Gehlhaar et al., 1995). Evolutionary algorithms can be combined with more traditional optimization techniques. This may be as simple as the use of a conjugate-gradient minimization used after primary search with an evolutionary algorithm (e.g., Gehlhaar et al., 1995), or it may involve simultaneous application of algorithms. There may also be a benefit to seeding an initial population with solutions derived from other procedures. Further, evolutionary computation can be used to optimize the performance of neural networks (Angeline et al., 1994), fuzzy systems (Haffner and Sebald, 1993), production systems (Wilson, 1995), and other program structures.

### e. Parallelism-

Evolution is a highly parallel process. As distributed processing computers become more readily available, there will be a corresponding increased potential for applying evolutionary algorithms to more complex problems. It is often the case that individual solutions can be evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel and only selection (which requires at least pair wise competition) requires some serial processing. In effect, the running time required for an application may be inversely proportional to the number of processors.

### f. Robust to Dynamic Changes –

Traditional methods of optimization are not robust to dynamic changes in the environment and often require a complete restart in order to provide a solution (e.g., dynamic programming). In contrast, evolutionary algorithms can be used to adapt solutions to changing circumstance. The available population of evolved solutions provides a basis for further improvement and in most cases it is not necessary, nor desirable, to reinitialize the population at random. Indeed, this procedure of adapting in the face of a dynamic environment can be used to advantage. The ability to adapt on the fly to changing circumstance is of critical importance to practical problem solving. For example, suppose that a particular simulation provides perfect fidelity to an industrial production setting. All workstations and processes are modeled exactly, and an algorithm is used to find a "perfect" schedule to maximize production. This perfect schedule will, however, never be implemented in practice because by the time it is brought forward for consideration, the plant will have changed: machines may have broken down, personnel may not have reported to work or failed to keep adequate records of prior work in progress, other obligations may require redirecting the utilization of equipment, and so forth. The "perfect" plan is obsolete before it is ever implemented. Rather than spend considerable computational effort to find such perfect plans, a better prescription is to spend less computational effort to discover suitable plans that are robust to expected anomalies and can be evolved on the fly when unexpected events occur.

## g) Capability for Self-Optimization-

Most classic optimization techniques require appropriate settings of exogenous variables. This is true of evolutionary algorithms as well. However, there is a long history of using the evolutionary process itself to optimize these parameters as part of the search for optimal solutions (Reed et al., 1967; Rosenberg, 1967; and others).

## h) Able to Solve Problems that have no known Solutions –

Perhaps the greatest advantage of evolutionary algorithms comes from the ability to address problems for which there are no human experts. Although human expertise should be used when it is available, it often proves less than adequate for automating problem-solving routines. Troubles with such expert systems are well known: the experts may not agree, may not be self-consistent, may not be qualified, or may simply be in error. Research in artificial intelligence has fragmented into a collection of methods and tricks for solving particular problems in restricted domains of interest. Certainly, these methods have been successfully applied to specific problems (e.g., the chess program Deep Blue). But most of these applications require human expertise. They may be impressively applied to difficult problems requiring great computational speed, but they generally do not advance our understanding of intelligence.

### 1.3.2 Swarm optimization advantages-

a) In PSO, the use of a Lagrangian or Cost function confers advantages similar to analysis using spectral /frequency domain methods. As the first response points out, the problem need not be differentiable or integrated of order 1.

b) The more obvious advantages are in the use of larger amounts of data though improvements in efficiency are limited given the equal asymptotic performance of other techniques.

c) Theoretically, the PSO allows improving on the ANN in terms of weeding out necessary local optima as the group's overall solution with each individual optimized datum.

## 1.4 Overview of evolutionary optimization algorithms-

Evolutionary computation[3] is an ambitious name for a simple idea: use the theory of evolution as an algorithm. Any program that uses the basic loop shown below could be termed evolutionary computation. Evolutionary algorithms operate on populations. The data structures would be chosen to represent the population, quality measures, and different ways to vary the data. It will need to decide how to tell when to stop. For any given problem there are many ways to implement an evolutionary computation system to attack the problem.

```
i)    Set i=0;
ii)   Generate the initial population P(i) at random;
iii)  REPEAT
        a) Evaluate the fitness of each individual in P(i);
        b) Select parents from P(i)based on their fitness;
        c) Apply search operators to the parents & produce generation P(i+1);
iv)   UNTIL the population converges & or the maximum time is reached
```

Evolutionary computation encompasses several major branches, i.e., evolution strategies, evolutionary programming, genetic algorithms and genetic programming, due largely to historical reasons. At the philosophical level, they differ mainly in the level at which they simulate evolution. At the algorithmic level, they differ mainly in their representations of potential solutions and their operators used to modify the solutions. From a computational point of view, representation and search are two key issues. Evolution strategies were first proposed by Rechenberg and Schwefel in 1965 as a numerical optimization technique. The original evolution strategy did not use populations. All evolutionary algorithms have two prominent features [4] which distinguish themselves from other search algorithms. First, they are all population-based. Second, there is communications and information exchange among individuals in a population. Such communications and information exchange are the result of selection and/or recombination in evolutionary algorithms.

## 1.5 Overview of swarm optimization algorithms-

Particle swarm optimization (PSO), in its historical version, is a collective, anarchic (in the original sense of the term), iterative method, with the emphasis on cooperation; it is partially random and without selection. Bird flocks, fish schools, and animal herds constitute representative examples of natural systems where aggregated behaviors are met, producing impressive, collision-free, and synchronized moves. In such systems, the behavior of each group member is based on simple inherent responses, although their outcome is rather complex from a macroscopic point of view. For example, the flight of a bird flock can be simulated with relative accuracy by simply maintaining a target distance between each bird and its immediate neighbors. This distance may depend on its size and desirable behavior. For instance, fish retain a greater mutual distance when swimming carefree, while they concentrate in very dense groups in the presence of predators. The groups can also react to external threats by rapidly changing their form, breaking in smaller parts and re-uniting, demonstrating a remarkable ability to respond collectively to external stimuli in order to preserve personal integrity. Similar phenomena are observed in physical systems. A typical example is the particle aggregation caused by direct attraction between particles due to Brownian motion or fluid shear. Humans too are characterized by agnate behaviors, especially at the level of social organization and belief formulation. However, these interactions can become very complex, especially in the belief space, where, in contrast to the physical space, the same point (a belief or an idea) can be occupied concurrently by large groups of people without collisions. The aforementioned aggregating behaviors characterized by the simplicity of animal and physical systems or the abstractness of human social behavior, intrigued researchers and motivated their further investigation through extensive experimentation and simulations (Heppner & Grenander, 1990; Reynolds, 1987; Wilson, 1975). The social sharing of information among individuals in a population can provide an evolutionary advantage. This general belief, which was suggested in several studies and supported by numerous examples from nature, constituted the core idea behind the development of PSO. Pseudo code of the operation of PSO is shown below –

Input: Number of particles **N**, swarm **S**, best positions **P.**

```
i)      Set t←0.
ii)     Initialize S and Set P=S.
iii)    Evaluate S and P, and define index g of the best position.
iv)     While (Termination criteria not met)
v)          Update S.
vi)         Evaluate S.
vii)        Update P and redefine index g.
viii)       Set t←t+1
ix)     End While
x)      Print best position found.
```

## 1.6 Hybridization and its advantage:-

Hybrid optimizations assume that one has implemented two or more algorithms for the same optimization. A hybrid optimization uses a heuristic to choose the best of these algorithms to apply in a given situation. Although evolutionary computation has been widely accepted for solving several important practical applications in engineering, business, commerce, etc., yet in practice sometimes they deliver only marginal performance. Inappropriate selection of various parameters, representation, etc. is frequently blamed. There is little reason to expect that one can find a uniformly best algorithm for solving all optimization problems.

This is in accordance with the *No Free Lunch theorem*[5], which explains that for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class. Let us consider, for m number of points, c be the cost value, f be the cost function, a1 & a2 be the two different algorithms. Now if we want to know, how F1, the set of '*f*' for which some algorithm '$a_1$' outperforms another algorithm 'a2' compare to F2, the set of '*f*'. To perform the comparison, the way of comparing sum over all '*f*' of P $(\vec{c}\,|$f, m, a1) to the sum over all 'f' of P $(\vec{c}\,|$f, m, a2). P $(\vec{c}\,|$f, m, a) is independent of 'a' when we average over all cost functions. So it can be shown as-

$$\sum_f \mathrm{P}\ (\vec{c}|\mathrm{f, m, a1}) = \sum_f \mathrm{P}\ (\underset{C}{\rightarrow}|\mathrm{f, m, a2})$$

Evolutionary algorithm behavior is determined by the exploitation and exploration relationship kept throughout the run. All these clearly illustrate the need for hybrid evolutionary approaches where the main task is to optimize the performance of the direct evolutionary approach. Recently, hybridization of evolutionary algorithms is getting popular due to their capabilities in handling several real world problems involving complexity, noisy environment, imprecision, uncertainty, and vagueness. As mentioned earlier, for several problems a simple Evolutionary algorithm might be good enough to find the desired solution. There are several types of problems where a direct evolutionary algorithm could fail to obtain a convenient (optimal) Solution. This clearly paves way to the need for hybridization of evolutionary algorithms with other optimization algorithms, machine learning techniques, heuristics etc. Some of the possible reasons for hybridization are as follows:-

i) To improve the performance of the evolutionary algorithm (example: speed of convergence)

ii) To improve the quality of the solutions obtained by the evolutionary algorithm.

iii) To incorporate the evolutionary algorithm as part of a larger system.

In case of evolutionary optimization, from initialization of population to the generation of offsprings, there are lots of opportunities to incorporate other techniques/algorithms etc. Population may be initialized by incorporating known solutions or by using heuristics, local search etc. Local search methods may be incorporated within the initial population members or among the offsprings. Evolutionary algorithms may be hybridized by using operators from other algorithms (or algorithms themselves) or by incorporating domain-specific knowledge. Evolutionary algorithm behavior is determined by the exploitation and exploration relationship kept throughout the run. Adaptive evolutionary algorithms have been built for inducing exploitation/exploration relationships that avoid the premature convergence problem and optimize the final results. The performances Of the evolutionary algorithm can be improved by combining problem-specific knowledge for particular problems.

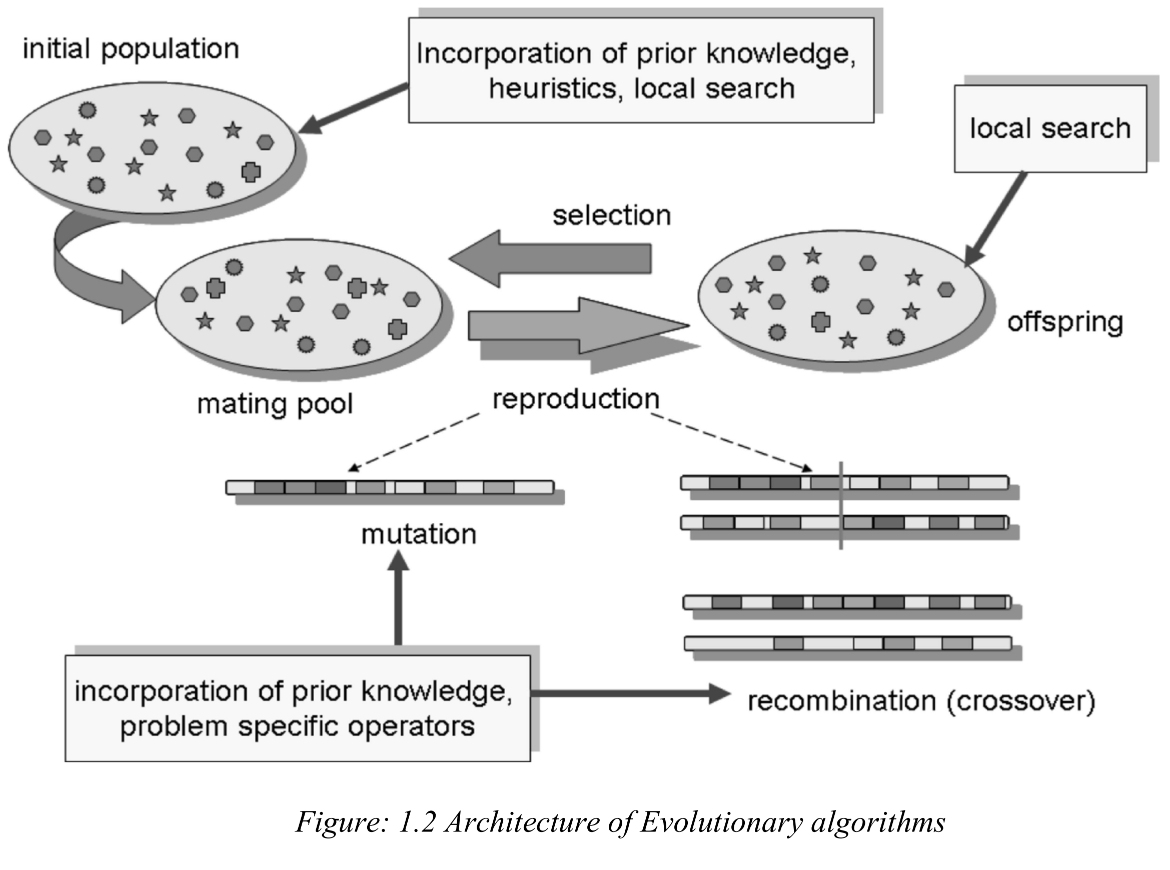

initial population
Incorporation of prior knowledge, heuristics, local search
local search
selection
offspring
mating pool
reproduction
mutation
incorporation of prior knowledge, problem specific operators
recombination (crossover)

*Figure: 1.2 Architecture of Evolutionary algorithms*

## 1.6.1 Architectures of Hybrid Evolutionary Algorithms:-

Several techniques and heuristics/metaheuristics have been used to improve the general efficiency of the evolutionary algorithm. Some of most used hybrid architectures are summarized as follows:-

1. Hybridization between an evolutionary algorithm and another evolutionary algorithm (example: a genetic programming technique is used to improve the performance of a genetic algorithm)
2. Neural network assisted evolutionary algorithms
3. Fuzzy logic assisted evolutionary algorithm
4. Particle swarm optimization (PSO) assisted evolutionary algorithm
5. Ant colony optimization (ACO) assisted evolutionary algorithm
6. Bacterial foraging optimization assisted evolutionary algorithm
7. Hybridization between evolutionary algorithm and other heuristics (such as local search, tabu search, simulated annealing, hill climbing, dynamic programming, greedy random adaptive search procedure, etc)

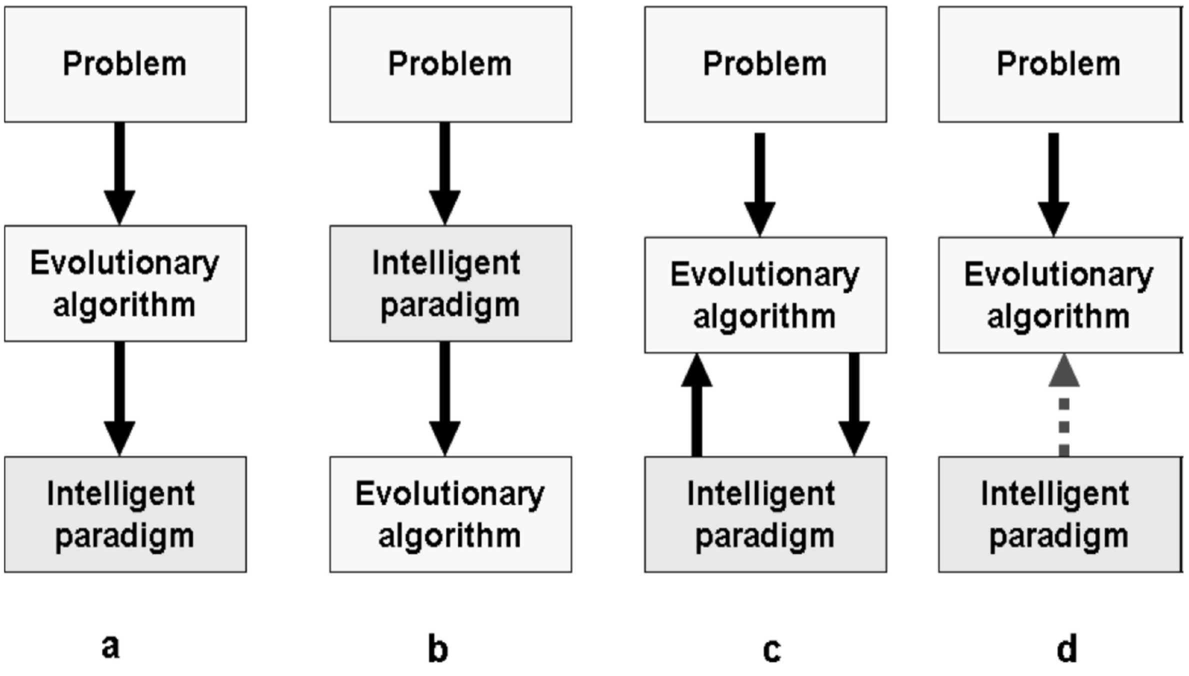


*Figure 1.3: Different steps of Evolutionary algorithm*

***References:-***

[1] Slawomir Koziel and Xin-She Yang (Eds.) ,‘Computational Optimization, Methods and Algorithms’. 2011 Springer-Verlag Berlin Heidelberg.

[2] David B. Fogel, ‘The Advantages of Evolutionary Computation’ Natural Selection, Inc.

[3] Daniel Ash lock, Department of Mathematics and Statistics University of Guelph, ‘Evolutionary Computation for Modeling and Optimization’. 2006 Springer Science+Business Media, Inc.

[4] Ruhul Sarker, Joarder Kamruzzaman, Charles Newton, ‘EVOLUTIONARY OPTIMIZATION (EvOpt): A BRIEF REVIEW AND ANALYSIS’ , International Journal of Computational Intelligence and Applications
Vol. 3, No. 4 (2003) 311-330, Imperial College Press

[5] David H. Wolpart, William G. Macready, ‘No free lunch theorem for search’,SFI-TR-95-02-010, The Santa Fe Institute, February 23,1996

# *CHAPTER-2*

## Differential Evolution

### 2.1 Introduction:-

Differential Evolution [1] was developed as a reliable and versatile global optimization tool for functions of many variables that is also easy to use. The first written publication on Differential Evolution appeared as a technical report in 1995. Since then, Differential Evolution has proven itself in competitions, such as IEEE's International Contest on Evolutionary Optimization (ICEO) in 1996 and 1997 and in the real world on a broad variety of applications. Like nearly all evolutionary algorithms (EAs), Differential Evolution is a population-based optimizer. It starts by sampling the objective function at multiple, randomly chosen initial points. The objective function f(X) is a function of d variables, $X = (X_0, X_1, ..., X_{d-1})$, $f(X) = f(X_0, X_1, ..., X_{d-1})$. The initial population has Np vectors X. The variable Np represents the number of points (vectors) in the population (it is not a product of N and P). Each vector $X_i$ is indexed with a number i from 0 to Np-1. Each vector represents an initial trial solution and the objective function is evaluated at each vector. The domain from which the Np initial vectors are chosen is defined by the preset parameter bounds. Like other population-based methods, Differential Evolution generates new points that are perturbations of existing points. Differential Evolution perturbs vectors with the scaled difference of two randomly selected population vectors. To produce a trial vector, Differential Evolution adds the scaled, random vector difference to a third randomly selected population vector. In the selection stage, the trial vector competes against the population vector of the same index. The vector with the better objective function value (e.g., lower for minimization) is marked as a member of the next generation. The procedure repeats until all Np population vectors have competed against a randomly generated trial vector. Once the last trial vector has been tested, the survivors of the Np pairwise competitions become parents for the next generation in the evolutionary cycle. It should be noted that in the operations research and numerical methods literature there is insight into how and why Differential Evolution works, including conditions for a convergence proof.

## 2.2Literature review:-

In an attempt[2] to find the global optimum of non-linear, non-convex, multi-modal and non-differentiable functions defined in the continuous parameter space (DRd), Storn and Price proposed the Differential Evolution (DE) algorithm in 1995. Since then, DE and its variants have emerged as one of the most competitive and versatile family of the evolutionary computing algorithms and have been successfully applied to solve numerous real world problems from diverse domains of science and technology. Unlike several other evolutionary computation techniques, basic DE stands out to be a very simple algorithm whose implementation requires only a few lines of code in any standard programming language. DE exhibits remarkable performance while optimizing a wide variety of objective functions in terms of final accuracy, computational speed, and robustness. Since 2010, apart from the combination of the existing mutation
schemes in DE, researchers have also made several attempts to devise new mutation schemes (involving difference vectors in various forms) which can provide improved search moves on complex fitness landscapes.

Below, it will be pointed out some of the reasons why the researchers have been looking at DE as an attractive optimization tool and as we shall proceed through this survey, these reasons will become more obvious.

1) With respect to other EAs, DE is much more simple and straightforward to implement. Main body of the algorithm takes few lines to code in any programming language. Simplicity to code is important for practitioners from other fields, since they may not be experts in programming and are looking for an algorithm that can be simply implemented and tuned to solve their domain-specific problems. Note that although PSO is also very easy to code, the performance of DE and its variants is largely better than the PSO variants over a wide variety of problems as has been indicated by studies like[3],[4].
2) As indicated by the recent studies on DE [3], [4], [5] despite its simplicity, DE exhibits much better performance in comparison with several others like G3 with PCX, MA-S2, ALEP, CPSO-H, and so on of current interest on a wide variety of problems including

unimodal, multimodal, separable, non-separable and so on. Although some very strong EAs like the restart CMAES was able to beat DE at CEC 2005 competition, on non-separable objective functions, the gross performance of DE in terms of accuracy, convergence speed, and robustness still makes it attractive for applications to various real-world optimization problems, where finding an approximate solution in reasonable amount of computational time is much weighted.

3) Control parameters are less (Cr, F, and NP DE). The effects of these parameters on the performance of the in classical algorithm are well studied. Simple adaptation rules for F and Cr have been devised to improve the performance of the algorithm to a large extent without imposing any serious computational burden [6], [7].
4) The space complexity of DE is low as compared to some of the most competitive real parameter optimizers like CMA-ES [S232]. This feature helps in extending DE for handling large scale and expensive optimization problems. Although CMA-ES remains very competitive over problems up to 100 variables, it is difficult to extend it to higher dimensional problems due to its storage, update, and inversion operations over square matrices with size the same as the number of variables.

In a recently published article [8], Neri and Tirronen reviewed a number of DE-variants for single-objective optimization problems and also made an experimental comparison of these variants on a set of numerical benchmarks. However, the article did not address issues like adapting DE to complex optimization environments involving multiple and constrained objective functions,noise and uncertainty in the fitness landscape, very large number of search variables, and so on.

## 2.2.1 DE in Different domain:-

This section reviews the extensions of DE for handling multiobjective, constrained, and large scale optimization problems. It also surveys the modifications of DE for optimization in dynamic and uncertain environments.

## *a. DE for Multiobjective Optimization:*

Due to the multiple criteria nature of most real-world problems especially in engineering, multi-objective optimization (MO) problems are ubiquitous. Multi-objective optimization problems involved multiple objectives, which should be optimized simultaneously and that often are in conflict with each other. This results in a group of alternative solutions, which must be considered equivalent in the absence of information concerning the relevance of the others. The concepts of dominance and Pareto optimality may be presented more formally in the following way. Consider without loss of generality the following multi-objective optimization problem with D decision variables x (parameters) and n objectives y:

Minimize $\vec{Y} = f(X) = (f1(x1, ...., x_D), ...., f_n(x1, ...., x_D))$

Where $\vec{X} = [x1, ....., x_D]^T \in P$ and $Y = [y1, ...., y_n]^T \in O$

and where $\vec{X}$ is called decision (parameter) vector, P is the parameter space, Y is the objective vector, and O is the objective space. A decision vector $\vec{A} \in P$ is said to dominate another decision vector $\vec{B} \in P$ (also written as $\vec{A} < \vec{B}$ for minimization) iff,

$$\forall i \in \{1, ...., n\} : fi(\vec{A}) \leq fi(\vec{B})$$
$$\wedge \exists j \in \{1, ....., n\} : fj(\vec{A}) < fj(\vec{B}).$$

Based on this convention, we can define non-dominated, Pareto-optimal solutions as follows.

Let $\vec{A} \in P$ be an arbitrary decision vector.

1) The decision vector A is said to be non-dominated regarding the set $P' \subseteq P$ if and only if there is no vector in P' which can dominate $\vec{A}$ .

2) The decision (parameter) vector A is called Pareto optimal if and only if $\vec{A}$ is non-dominated regarding the whole parameter space P .

Many evolutionary algorithms were formulated by the researchers to tackle multi-objective problems in recent past. Kukkonen and Lampinen extended DE/rand/1/bin to solve multi-objective optimization problems in their approach called generalized differential evolution (GDE). To deal with the shortcomings of GDE2 regarding slow convergence, Kukkonen and Lampinen proposed an improved version called GDE3 [9] (a combination of the earlier GDE versions and the Pareto-based differential evolution algorithm).

This version added a growing population size and non-dominated sorting (as in the NSGAII ) to improve the distribution of solutions in the final Pareto front and to decrease the sensitivity of the approach to its initial parameters. Santana-Quintero and Coello Coello proposed the ∈-MyDE in [10]. This approach keeps two populations: the main population (which is used to select the parents) and a secondary (external) population, in which the concept of ∈-dominance is adopted to retain the non-dominated solutions found and to distribute them in a uniform way. This approach keeps two populations: the main population (which is used to select the parents) and a secondary (external) population, in which the concept of ∈-dominance is adopted to retain the non-dominated solutions found and to distribute them in a uniform way. In [11], Xue et al. came up with the multiobjective DE (MODE) in which the best individual is adopted to create the offspring. A Pareto-based approach is introduced to implement the selection of the best individual. If a solution is dominated, a set of non-dominated individuals can be identified and the "best" turns out to be any individual (randomly picked) from this set.Robic and Filipic presented a DE for multiobjective optimization (called DEMO) in [12]. This algorithm combines the advantages of DE with the mechanisms of Pareto-based ranking and crowding distance sorting.

DEMO only maintains one population and it is extended when newly created candidates take part immediately in the creation of the subsequent candidates. This enables a fast convergence toward the true Pareto front, while the use of non-dominated sorting and crowding distance (derived from the NSGA-II) of the extended population promotes the uniform spread of solutions. Iorio and Li [13] proposed the non-dominated sorting DE (NSDE), which is a simple modification of the NSGA-II . The only difference between this approach and the NSGA-II is in the method for generating new individuals. The NSGA-II uses a real-coded crossover and mutation operator, but in the NSDE, these operators were replaced with the operators of differential evolution. NSDE was shown to outperform NSGA-II on set of rotated MO problems with strong interdependence of variables.

Some researchers have proposed approaches that use non- Pareto based multiobjective concepts like combination of functions, problem transformation, and so on. For example, Babu and Jehan [14] proposed a DE algorithm for MO problems, which uses the DE/rand/1/bin variant with two different mechanisms to solve bi-objective problems: first, incorporating one objective function as a constraint, and secondly using an aggregating function. Li and Zhang, [15] proposed a multiobjective differential evolution algorithm based on decomposition (MOEA/D-DE) for continuous multiobjective optimization problems with variable linkages. The DE/rand/1/bin scheme is used for generating new trial solutions, and a neighborhood relationship among all the sub-problems generated is defined, such that they all have similar optimal solutions. In [15], they introduce a general class of continuous MO problems with complicated Pareto set (PS) shapes and reported the superiority of MOEA/D-DE over NSGA-II with DE type reproduction operators. Summation of normalized objective values with diversified selection approach was used in [16] without the need for performing non-dominated sorting.

## *b. DE for Constrained Optimization:*

Most of the real world optimization problems involve finding a solution that not only is optimal, but also satisfies one or more constraints. A general formulation for constrained optimization may be given in the following way.

Find $\vec{X} = [x_1, x_2, ..., x_D]^T$ $\vec{X}$ □$\Re^D$

to minimize: $f(\vec{X})$

subjected to inequality constraints: $gi(\vec{X}) \leq 0 \quad i=1,2,....,K$

equality constraints: $hj(\vec{X}) = 0 \quad j=1,2,....,N$

and boundary constraints: $x_{j,min} \leq x_j \leq x_{j,max}$.

Boundary constraints are very common in real-world applications, often because parameters are related to physical components or measures that have natural bounds, e.g., the resistance of a wire or the mass of an object can never be negative. In order to tackle boundary constraints, penalty methods drive solutions from restricted areas through the action of an objective function-based criterion. DE uses the following four kinds of penalty method to handle boundary constraint violation.

1) Brick wall penalty [17]: if any parameter of a vector falls beyond the pre-defined lower or upper bounds, objective function value of the vector is made high enough (by a fixed big number) to guarantee that it never gets selected.

2) Adaptive penalty [18], [19]: similar to brick wall penalty, but here the increase in the objective function value of the offender vector may depend on the number of parameters violating bound constraints and their magnitudes of violation.

3) Random re-initialization [21], [17]: replaces a parameter that exceeds its bounds by a randomly chosen value from within the allowed range following (1).

4) Bounce-back [17]: relocates the parameter in between the bound it exceeded and the corresponding parameter from the base vector.

The first known extension of DE toward the handling of inequality constrained optimization problems (mainly design centering) was by R. Storn [21]. He proposed a multimember DE (called CADE: constraint adaptation with DE, in his paper) that generates M (M > 1) children for each individual with three randomly selected distinct individuals in the current generation, and then only one of the M + 1 individuals will survive in the next generation. Mezura-Montes et al. used the concept also to solve constrained optimization problems. Zhang et al. [22] mixed the dynamic stochastic ranking with the multimember DE framework and obtained promising performance on the 22 benchmarks taken from the CEC 2006 competition on constrained optimization [23]. some studies have also been reported regarding parameter control in DE for constrained optimization. Brest et al. have proposed an adaptive parameter control for two DE parameters related to the crossover and mutation operators. Huang et al. [24] used an adaptive mechanism to select among a set of DE variants to be used for the generation of new vectors based on a success measure. Moreover, they also adapted some DE parameters to control the variation operators. Very recently Mezura-Montes and Palomeque-Ortiz presented the adaptive parameter control in the diversity differential evolution (DDE) algorithm for constrained optimization. Three parameters namely the scale factor F, the crossover rate Cr, and the number of offspring generated by each target vector NO, are self-adapted by encoding them within each individual and a fourth parameter called selection ratio $S_r$ is controlled by a deterministic approach.

## *C. DE for Large-Scale Optimization:*

In last few decades, several type of bio-inspired optimization algorithms have been designed and applied to solve optimization problems. Although these approaches have shown excellent search abilities when applied to some 30–100 dimensional problems, usually their performance deteriorates quickly as the dimensionality of search space increases beyond 500. The reasons appear to be two-fold. First, complexity of the problem usually increases with the size of problem, and a previously successful search strategy may no longer be capable of finding the optimal solution. Second, the solution space of the problem increases exponentially with the problem size, and a more efficient search strategy is required to explore all the promising regions in a given time budget. Since the performance of basic DE schemes also degrade with massive increase in problem dimensions, some important attempts have In [25], Noman and Iba proposed fittest individual refinement (FIR), a crossover based local search method for DE, such that the FIR scheme accelerates DE by enhancing its search capability through exploration of the neighborhood of the best solution in successive generations. The proposed memetic version of DE (augmented by FIR) was shown to obtain an acceptable solution with a lower number of evaluations particularly for higher dimensional functions. Another memetic DE for high-dimensional optimization was presented by Gao and Wang [26], where the stochastic properties of chaotic system is used to spread the individuals in search spaces as much as possible and the simplex search method is employed to speed up the local exploiting and the DE operators help the algorithm to jump to a better point. been made by the researchers to make DE suitable for handling such large-scale optimization problems.

In terms of optimizing high-dimensional problems, cooperative co-evolution (first proposed by Potter and De Jong for GAs ) with the following divide-and-conquer strategy has proven an effective choice.

1) Problem decomposition: splitting the object vectors into some smaller subcomponents.

2) Optimize subcomponents: evolve each subcomponent with a certain optimizer separately.

3) Cooperative combination: combine all subcomponents to form the whole system.

In [27], the authors proposed two DE-variants (DECCI and DECC-II) that use self-adaptive NSDE (SaNSDE) (a synergy of the works reported in [28] and [7]) in a cooperative co-evolutionary framework with novel strategies for problem decomposition and subcomponents' cooperation. The algorithms were tested on a set of widely used benchmarks scaled up to 500 and 1000 dimensions. An important extension of the same work for better performance on rotated and non-separable high-dimensional functions has been reported in [29] where the authors use random grouping scheme with adaptive weighting for problem decomposition and coevolution The theoretical analysis illustrates how such strategies can help to capture variable interdependencies in non-separable problems. Recently, Parsopoulos devised a cooperative micro- DE, which employs small cooperative subpopulations (with only few individuals) to detect subcomponents of the original problem's solution concurrently. The subcomponents are combined through cooperation of subpopulations to build complete solutions of the problem. Zamuda et al. proposed a DE-variant for large scale global optimization, where original
DE is extended by log-normal self-adaptation of its control parameters and combined with cooperative co-evolution as a dimension decomposition mechanism.Su presented a surrogate assisted DE framework based on Gaussian process for solving large-scale computationally expensive problems in. Brest et al. [6] investigated a self-adaptive DE (abbreviated as jDEdynNP-F) where control parameters F and Cr are self-adapted and a population-size reduction method is used. The proposed jDEdynNP-F algorithm also employs a mechanism for sign changing of F with some probability based on the fitness values of randomly chosen vectors, which are multiplied by F in the mutation step of DE. The algorithm achieved third rank in CEC 2008 special session and competition on high-dimensional real-parameter optimization [30] that included non-separable functions.

## *D. DE for Optimization in Dynamic and Uncertain Environments*

In many real world applications, EAs often have to deal with optimization problems in the presence of a wide range of uncertainties. In general, there are four ways in which uncertainty may creep into the computing environment .
First, the fitness function may be noisy. Second, the design variables and/or the environmental parameters may change after optimization, and the quality of the obtained optimal solution should be robust against environmental changes or

deviations from the optimal point. Third, the fitness function may be approximated, which means that the fitness function suffers from approximation errors. Finally, the optimum of the problem to be solved changes its location over time and, thus, the optimizer should be able to track the optimum continuously. In all these cases, the EAs should be equipped with additional measures so that they are still able to work satisfactorily.

For a noisy problem, a deterministic choice of the scale factor and the greedy selection methods can be inadequate and a standard DE can easily fail at handling a noisy fitness function, as experimentally shown in [31]. Looking at the problem from a different perspective, the DE employs too much deterministic search logic for a noisy environment and therefore tends to stagnate. Das et al. [32] made an attempt to improve the performance of DE on noisy functions by first varying the scale factor randomly between 0.5 and 1 and secondly by incorporating two not-so-greedy selection mechanisms (threshold based selection and stochastic selection) in DE. Liu et al. [33] combined the advantages of the DE algorithm, the optimal computing budget allocation technique and simulated annealing (SA) algorithm to devise a robust hybrid DE method abbreviated as DEOSA) that can work well in noisy environments. Mendes and Mohais presented DynDE [34]—a multi-population DE algorithm, developed specifically to optimize slowly time-varying objective functions. DynDE does not need any parameter control strategy for the F or Cr. The main components in DynDE are as follows.

1) Usage of several populations in parallel.

2) Usage of uniform dither for F? [0, 1] as well as Cr $\in$ [0, 1].

3) To maintain diversity of the population based on two approaches.

a) Re-initialization of a population if the best individual of a population gets too close to the best individual of another population. The population with the absolute best individual is kept while the other one is reinitialized. This way the various populations are prevented from merging.

b) Randomization of one or more population vectors by adding a random deviation to the components.

*E. DE for Multimodal Optimization and Niching:*

Many practical objective functions are highly multimodal and likely to have several high quality global and/or local solutions. Often, it is desirable to identify as many of these solutions as possible so that the most appropriate solution can
be chosen. In order to identify many solutions of a multimodal optimization problem, several "niching" techniques have been developed. A niching method empowers an EA to maintain multiple groups within a single population in order to locate different optima. The niching techniques include crowding [35], fitness sharing, clearing, restricted tournament selection [36],
The crowding method allows competition for limited resources among similar individuals, i.e., within each niche. Generally, the similarity is measured using distance between individuals. The method compares an offspring with some randomly sampled individuals from the current population. The most similar individual will be replaced if the offspring is superior. Thomsen extended DE with a crowding scheme named as crowding DE (CDE) [35] to solve multimodal optimization problems. In CDE, when an offspring is generated, it will only compete with the most similar (measured by Euclidean distance) individual in the population. The offspring will replace this individual if it has a better fitness value.
The fitness sharing method divides the population into different subgroups according to parameter space similarity of the individuals. An individual must share its information with other individuals within the same niche. The shared fitness for ith individual can be represented by using the following equation:

$$f_{\text{shared}}(i) = \frac{f_{original}(i)}{\sum_{j=1}^{N} sh(d_{ij})}$$

where the sharing function is calculated as:

$$sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\sigma_{share}})^{\alpha} & \text{if } d_{ij} < \sigma_{share} \\ 0 & otherwise \end{cases}$$

and d$_{ij}$ is the distance between individuals $i$ and $j$, $\sigma_{share}$ is the sharing radius, N is the population size and α is a constant called sharing level. Thomsen integrated the fitness sharing concept with DE to form the sharing DE.

## 2.3 Main steps of DE:

When we try to optimize a system, we aim at finding out such a set of values of the system parameters for which the overall performance of the system will be the best under some given conditions. Usually, the parameters governing the system performance are represented in a vector like – $\vec{X} = [x_1, x_2, \ldots, x_D]^T$ For real parameter optimization, as the name implies, each parameter $x_i$ is a real number. To measure how far the "best" performance we have achieved, an objective function (or fitness function) is designed for the system. The task of optimization is basically a search for such the parameter vector $\vec{X}^*$ which minimizes such an objective function $f(\vec{X})(f : \Omega \subseteq \Re^D \rightarrow \Re)$ , i.e., $f(\vec{X}^*) < f(\vec{X})\ \forall\ \vec{X} \in \Omega$, where $\Omega$ is a non-empty large finite set serving as the domain of the search. For unconstrained optimization problems $\Omega = \Re^D$ Since, max $\{f(\vec{X})\}$ = - min$\{- f(\vec{X})\}$ , the restriction to minimization is without loss of generality. In general, the optimization task is complicated by the existence of non- linear objective functions with multiple local minima. A local minimum $f_l = f(\overrightarrow{Xl})$ may be defined as $\exists \varepsilon > 0\ \forall\ \vec{X} \in \Omega$ : $\|\vec{X} - \vec{X}l\| < \varepsilon \Rightarrow f_l \leq f(\vec{X})$, where $\|.\|$ indicates any p-norm distance measure.

DE is a simple real parameter optimization algorithm. It works through a simple cycle of stages, presented in below figure.

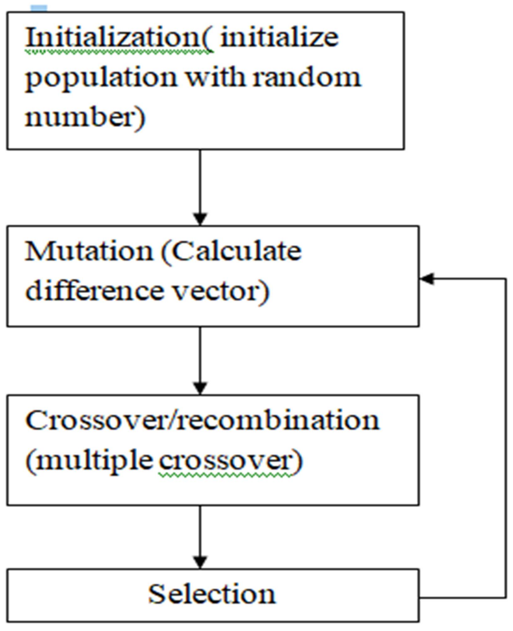

Initialization( initialize population with random number)
Mutation (Calculate difference vector)
Crossover/recombination (multiple crossover)
Selection

*Fig2.1: Differential Evolution Stages*

## *A. Initialization of the Parameter Vectors :-*

DE searches for a global optimum point in a *D*-dimensional real parameter space $\Re^D$ . It begins with a randomly initiated population of NP D dimensional real-valued parameter vectors. Each vector, also known as genome/chromosome, forms a candidate solution to the multidimensional optimization problem. We shall denote subsequent generations in DE by G = 0, 1..., $G_{max}$. Since the parameter vectors are likely to be changed over different generations, we may adopt the following notation for representing the *i*th vector of the population at the current generation:

$$X_{i,G} = [X_{1,i,G}, X_{2,i,G}, X_{3,i,G}, \ldots.., X_{D,i,G}]$$

For each parameter of the problem, there may be a certain range within which the value of the parameter should be restricted, often because parameters are related to physical components or measures that have natural bounds (for example if one parameter is a length or mass, it cannot be negative). The initial population (at G = 0) should cover this range as much as possible by uniformly randomizing individuals within the search space constrained by the prescribed minimum and maximum bounds: $\vec{X}_{min} = \{x_{1,min}, x_{2,min}, \ldots, x_{D,min}\}$ and Xmax = $\{x_{1,max}, x_{2,max}, \ldots, x_{D,max}\}$. Hence we may initialize the *j*th component of the *i*th vector as

$$x_{j,i,0} = x_{j,min} + rand_{i,j}[0, 1] \cdot (x_{j,max} - x_{j,min})$$

where randi,j[0, 1] is a uniformly distributed random number lying between 0 and 1 (actually 0 ≤ *randi,j*[0, 1] ≤ 1) and is instantiated independently for each component of the i-th vector.

## *A. Mutation with Difference Vectors*:-

Biologically, “mutation” means a sudden change in the gene characteristics of a chromosome. In the context of the evolutionary computing paradigm, however, mutation is also seen as a change or perturbation with a random element. In DE-literature, a parent vector from the current generation is called target vector, a mutant vector obtained through the differential mutation operation is known as donor vector and finally an offspring formed by recombining the donor with the target vector is called trial vector. In one of the simplest forms of DE-mutation, to create the donor vector for each $i^{th}$ target vector from the current population, three other distinct parameter vectors, say $\vec{X}\, r_1^i$ , $\vec{X}\, r_2^i$, $\vec{X}\, r_3^i$ are sampled randomly from the current population. The indices $r_{1i}$ , $r_{2i}$ , and $r_{3i}$ are mutually exclusive integers randomly chosen from the range [1, NP] which are also different from the base vector index *i*. These indices are randomly generated once for each mutant vector. Now the difference of any two of these three vectors is scaled by a scalar number F (that typically lies in the interval [0.4, 1]) and the scaled difference is added to the third one whence we obtain the donor vector $V_{i,G}$. We can express the process as

$$V_{i,G} = X\, r_1^i + F \cdot (X\, r_2^i{}_{,G} - X\, r_3^i{}_{,G}).$$

The process is illustrated on a 2-D parameter space (showing constant cost contours of an arbitrary objective function) in below Fig. 2.2.

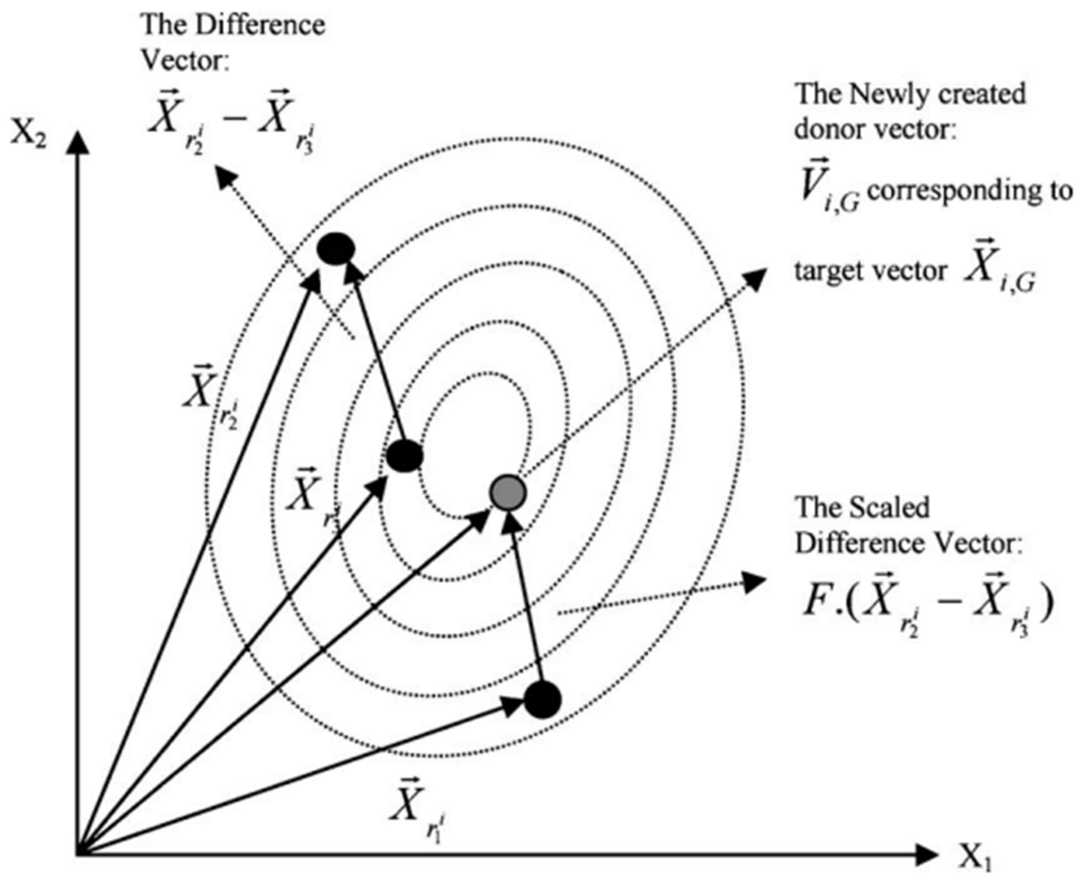

The Difference Vector: $\vec{X}_{r_2^i} - \vec{X}_{r_3^i}$
The Newly created donor vector: $\vec{V}_{i,G}$ corresponding to target vector $\vec{X}_{i,G}$
$X_2$
$\vec{X}_{r_2^i}$
$\vec{X}_{r_3^i}$
The Scaled Difference Vector: $F.(\vec{X}_{r_2^i} - \vec{X}_{r_3^i})$
$\vec{X}_{r_1^i}$
$X_1$

*Figure 2.2[2]: Illustrating a simple DE mutation scheme in 2-D parametric space.*

## *A. Crossover:-*

To enhance the potential diversity of the population, a crossover operation comes into play after generating the donor vector through mutation. The donor vector exchanges its components with the target vector $\vec{X}_{i,G}$ under this operation to form trial vector $\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, \ldots, u_{D,i,G}]$.The DE family of algorithms can use two kinds of crossover methods—exponential (or two-point modulo) and binomial (or uniform) [17]. In exponential crossover, we first choose an integer n randomly among the numbers [1, D]. This integer acts as a starting point in the target vector, from where the crossover or exchange of components with the donor vector starts. We also choose another integer L from the interval [1, D]. L denotes the number of components the donor vector actually contributes to the target vector. After choosing n and L the trial vector is obtained as

$u_{j,i,G} = v_{j,i,G}$ for j = $\langle n \rangle_D$ $\langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D$

$x_{j,i,G}$ for all other j ∈ [1, D]

where the angular brackets D denote a modulo function with modulus D. The integer L is drawn from [1, D] according to the following pseudo-code:

```
L = 0; DO
    {
    L = L + 1;
    } WHILE ((rand(0, 1) ≤ Cr) AND (L ≤ D)).
```

"$C_r$" is called the crossover rate and appears as a control parameter of DE just like F. Hence in effect, probability (L = υ) = (Cr)υ - 1 for any positive integer v lying in the interval [1, D]. For each donor vector, a new set of n and L must be chosen randomly as shown above.
On the other hand, binomial crossover is performed on each of the D variables whenever a randomly generated number between 0 and 1 is less than or equal to the Cr value. In this case, the number of parameters inherited from the donor has a (nearly) binomial distribution. The scheme may be outlined as

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & if(randi,j\ [0,1] \le Cr\ or\ j = jrand \\ x_{j,i,G} & Otherwise \end{cases}$$

where, as before, $\text{rand}_{i,j}[0, 1]$ is a uniformly distributed random number, which is called anew for each jth component of the ith parameter vector. $j_{rand} \in [1, 2, ...., D]$ is a randomly chosen index, which ensures that $\vec{U}_{i,G}$ gets at least one component from $\vec{V}_{i,G}$. It is instantiated once for each vector per generation. We note that for this additional demand, $C_r$ is only approximating the true probability $P_{Cr}$ of the event that a component of the trial vector will be inherited from the donor.

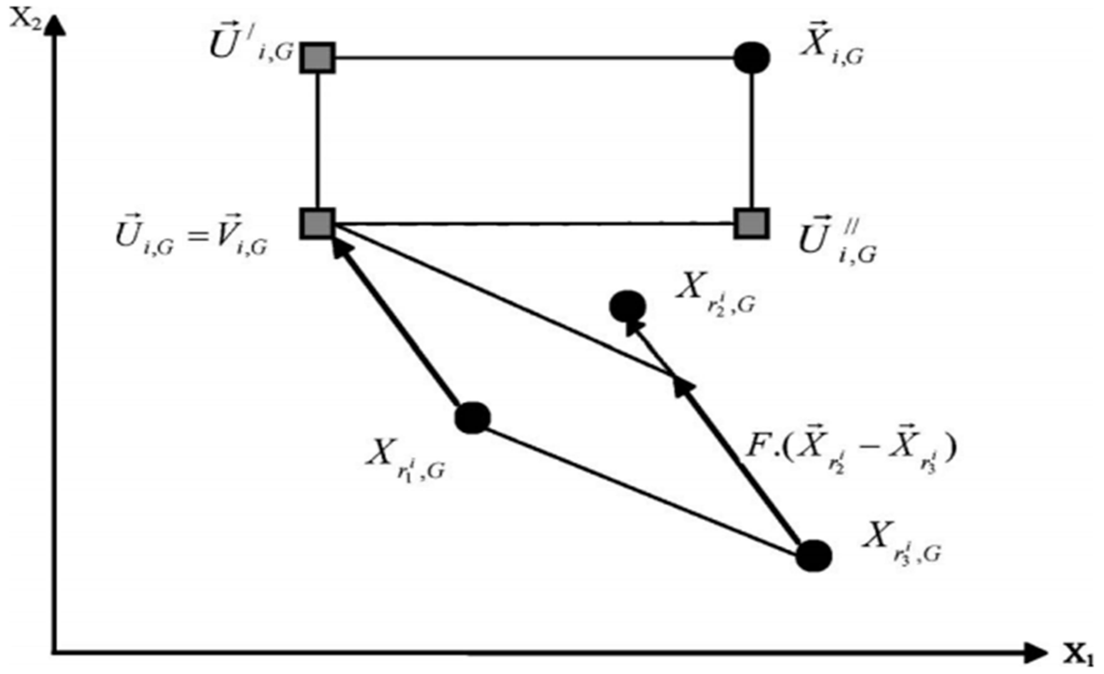


*Fig.2.3[2]: Different possible trial vectors formed due to uniform/binomial crossover between the target and the mutant vectors in 2-D search space.*

Also, one may observe that in a 2-D search space, three possible trial vectors may result from uniformly crossing a mutant/donor vector $\vec{V}_{i,G}$ with the target vector $\vec{X}_{i,G}$. These trial vectors are as follows.

1) $\vec{U}_{i,G} = \vec{V}_{i,G}$ such that both the components of $\vec{U}_{i,G}$ are inherited from $\vec{V}_{i,G}$.

2) $\overrightarrow{U^{/}}_{i,G}$ , in which the first component (j = 1) comes from $\vec{V}_{i,G}$ and the second one (j = 2) from $\vec{X}_{i,G}$.

3) $\overrightarrow{U^{//}}_{i,G}$ , in which the first component (j = 1) comes from $X_{i,G}$ and the second one ($j$ = 2) from $V_{i,G}$.

The possible trial vectors due to uniform crossover are illustrated in above Fig.

D) *Selection:-*

To keep the population size constant over subsequent generations, the next step of the algorithm calls for selection to determine whether the target or the trial vector survives to the next generation, i.e., at G = G + 1. The selection operation is described as

$$\vec{X}_{i,G+1} = \vec{U}_{i,G} \quad \text{if} \quad f(\vec{U}_{i,G}) \le f(\vec{X}_{i,G})$$
$$= \vec{X}_{i,G} \quad \text{if} \quad f(\vec{U}_{i,G}) > f(\vec{X}_{i,G})$$

where f (X ) is the objective function to be minimized. Therefore, if the new trial vector yields an equal or lower value of the objective function, it replaces the corresponding target vector in the next generation; otherwise the target is retained in the population. Hence, the population either gets better (with respect to the minimization of the objective function) or remains the same in fitness status, but never deteriorates. In above equation, the target vector is replaced by the trial vector even if both yields the same value of the objective function—a feature that enables DE-vectors to move over.

## 2.4 Pseudo Code:

**Step 1:** Read values of the control parameters of DE: scale factor F, crossover rate Cr, and the population size NP from user.

**Step 2**: Set the generation number G = 0 and randomly initialize a population of NP individuals $P_G = \{\vec{X}_{i,G}, ......, \vec{X}_{N,P,G}\}$ with $\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, ....., x_{D,i,G}]$ and each individual uniformly distributed in the range $[\vec{X}_{min}, \vec{X}_{max}]$, where $\vec{X}_{min} = \{x_{1,min}, x_{2,min}, ..., x_{D,min}\}$ and $\vec{X}_{max} = \{x_{1,max}, x_{2,max}, ..., x_{D,max}\}$ with i = [1, 2, ...., NP].

**Step 3.** WHILE the stopping criterion is not satisfied

```
DO
FOR i = 1 to NP //do for each individual sequentially
```

### Step 3.1 Mutation Step

Generate a donor vector $V_{i,G} = \{v_{1,i,G}, ......., \}$
$\{v_{D,i,G}\}$ corresponding to the ith target vector
$\vec{X}_{i,G}$ via the differential mutation scheme of DE
as: $\vec{V}_{i,G} = \vec{X}r^{i}_{1,G} + F . (\vec{X}r^{i}_{2,G} - \vec{X}r^{i}_{3,G})$

### Step 3.2 Crossover Step

Generate a trial vector $\vec{U}_{i,G} = \{u_{1,i,G}, ......., u_{D,i,G}\}$ for the ith target vector $\vec{X}_{i,G}$ through binomial crossover in the following way: $u_{j,i,G} = v_{j,i,G}$, if $(rand_{i,j}[0, 1] \leq C_r$ or $j = j_{rand})$ $x_{j,i,G}$, otherwise,

### Step 3.3 Selection Step

```
Evaluate the trial vector Ui,G
IF f (Ui,G) ≤ f (X i,G), THEN X i,G+1 = Ui,G
ELSE X i,G+1 = X i,G.
END IF
END FOR
Step 3.4 Increase the Generation Count
G = G + 1
END WHILE
```

## 2.5 Few Real World Applications:

### *2.5.1 OPTIMIZATION OF A G992.1(an International Telecommunication Union standard):*

The first problem is a design problem by Infineon AG which was first reported in [37]. The ITU-standard G992.1 [38] for asymmetrical digital subscriber lines (ADSL) uses a two pair twisted line to transport upstream and downstream data simultaneously from/into the home of a subscriber to achieve a data rate of 6.144Mbps in downstream (DS) and 640kbps in upstream (US). There are two annexes, Annex A which defines ADSL(Asymmetric digital subscriber line) over Plain Old Telephone Service (POTS) and Annex B which defines ADSL over Integrated Services Digital Network (ISDN). For the latter the frequency bands of interest are sketched in below Fig.2.4

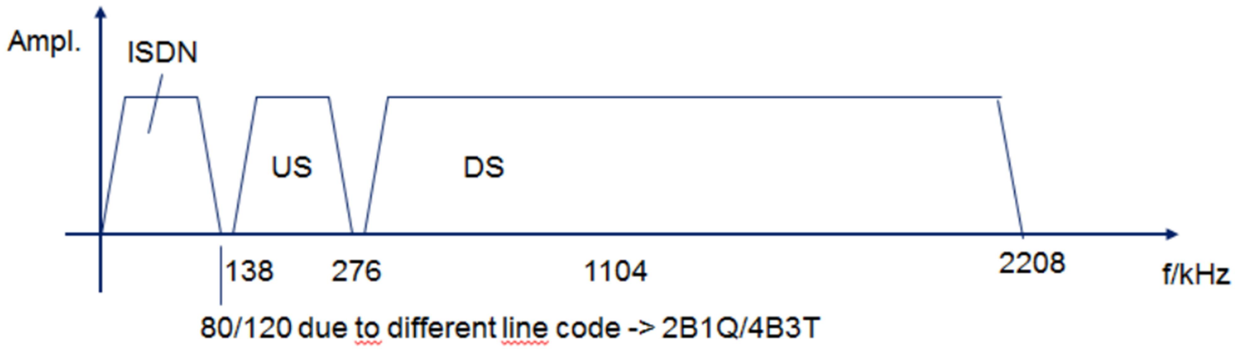


*Fig.2.4: Frequency Bands of a signal on Integrated Services Digital Network (ISDN)*

The challenge in this configuration was to minimize the interference between the adjacent bands and hence achieve a maximum data rate for all the signals ISDN, US, and DS. The main part of the electrical circuit for the analog frontend using
the Infineon Geminax ® chipset is shown in below Fig.

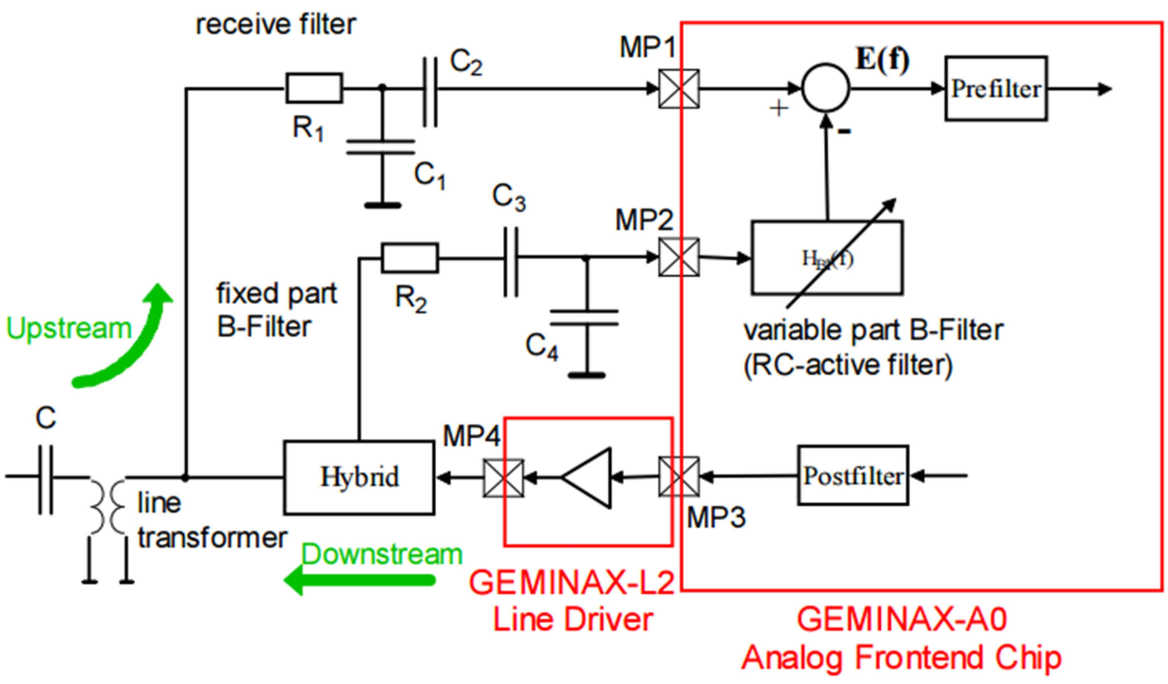


*Fig 2.5 : Analog frontend circuitry belonging to Infineon line cards with the GEMINAX ® ADSL chipset*

When a DS signal is sent via the GEMINAX-L2 line driver it will also enter the receive path for US via the lowpass circuitry consisting of $R_1$, $C_1$, and $C_2$. Even though the band of the DS signal is different from the US band there are still DS sidelobes acting as disturbers which reduce the signal-to-noise rate (SNR) in the US and therefore the achievable US rate. In order to mitigate the DS echo E(f) the echo-cancelling circuitry consisting of R2, C3, and C4, and the RC-active balancing filter $H_{BI}(f)$ is employed. Since $H_{BI}(f)$ is an active component it also injects amplifier noise into the US band. The design task was therefore to minimize the echo E(f) as well as the amplifier noise of $H_{BI}(f)$ simultaneously which turned this design problem into a two-objective optimization problem. The design was further complicated by several constraints:
*Firstly,* not only the DS side lobes disturb the US band but also the ISDN side lobes coming from the lower part of the spectrum. Hence a tolerance scheme for ISDN signal suppression had also to be mandated.
*Secondly,* the measurement point MP2 was subject to the peak voltage limit of 1.26V which was not allowed to be exceeded in order to prevent nonlinear behavior of the GEMINAX-A0.
*Thirdly,* the maximum values for the resistors Ri were limited to 100kΩ and the capacitors Ci were limited to 2nF which was a result of component size limitations due to an already existing printed circuit board layout.
*Fourthly,* all capacitors had to exhibit discrete values from the E12-Series and the resistors from the E-96 series [40] in order to reduce component costs. From the description it is clear that this problem is multiobjective, constrained and of mixed integer type so it is not surprising that the analog design engineers failed to come up with a satisfactory result which meets all constraints after many weeks of design activity.

### *2.5.2 GAUSSIAN FILTER DESIGN:*

The next real-world example is a digital filter design used at Rohde & Schwarz in an FPGA-based channel simulator for wireless applications [41] [42], [43]. The tolerance scheme for the magnitude of the filter follows a gaussian shape exhibiting an extremely narrow bandwidth in the range $\Omega \in [0, 0.0046]$. The filter had to be implemented by an FPGA-based IIR-filter of 8th order and a word-length of 32 bits. For ease of implementation and minimization of digital quantization noise the filter structure was split into four biquad stages [44]. The magnitude at $\Omega = 0.0046$ was required to be -57dB. All endeavors to use standard filter design tools built into MATLAB ® [44] failed due to strong violations of the tolerance scheme when the coefficients were finally set to their finite word-length of 32 bits. This failure was due to the two-step approach taken by the tools where the coefficients were determined with infinite precision in the first step and quantization in the second. So a design using Differential Evolution (DE) [45], [46] had to come to the rescue where the coefficient quantization was taken directly into account.

### *2.5.3 PLL DESIGN:*

For the VTC video tester family of Rohde&Schwarz ® a phase locked loop (PLL) was needed to assist the measurement of eye diagrams [46]. The reference frequency, which serves as the input to the PLL, can vary between 25MHz and 340MHz, depending on the video stream. The PLL was required to generate a frequency of 7.15GHz with a constant bandwidth of 4MHz. In order to keep the bandwidth constant the gain of the amplification inside the PLL needed to be adjusted by pertinently choosing a gain dependent resistor $R_{ideal,j}$ where j is associated with a gain vj. $R_{ideal,j}$ is approximated by a parallel circuit of up to 12 resistors that are combined by the three switches ADG1204YRUZ shown in the middle of Fig 2.6

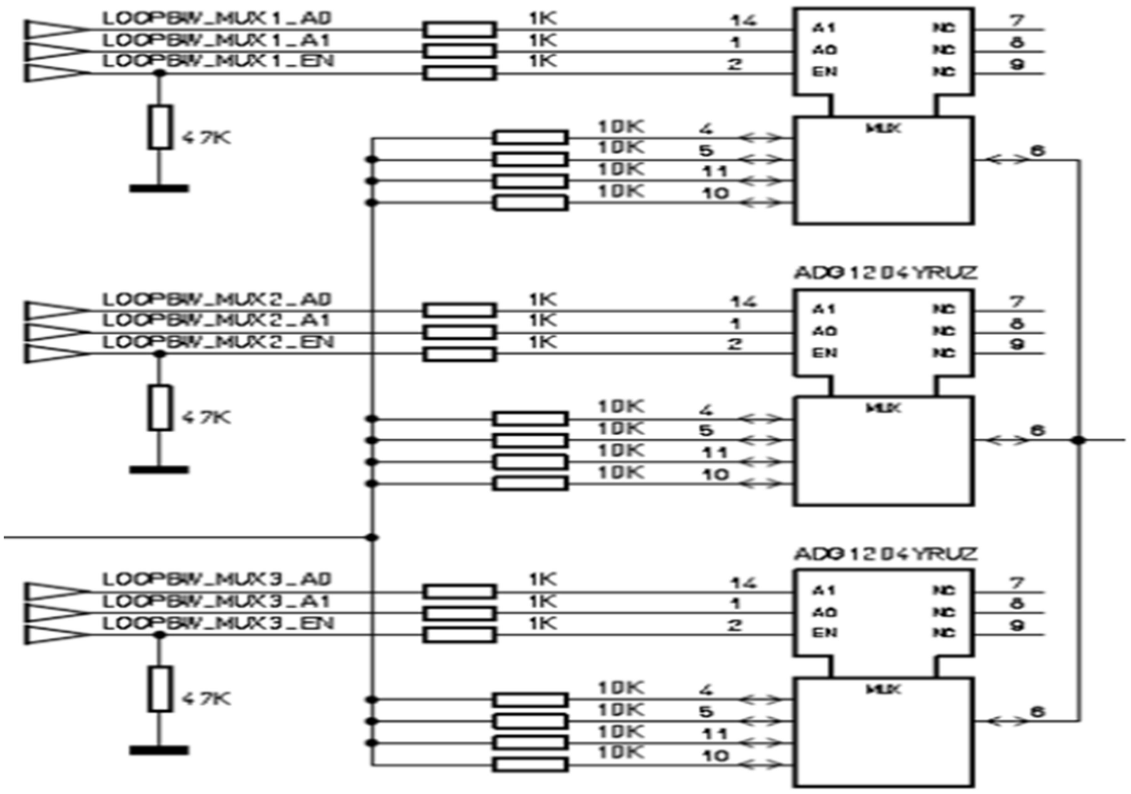


*Fig 2.6. PLL section of the VTC video tester family of Rohde & Schwarz ® with the three switches ADG1204YRUZ that implement a variable gain determining resistor by a parallel circuit of up to 12 resistors*

The switches have five positions in order for each switch to add 0, 1, 2, 3, or 4 resistors to the combination which results in $5^3$=125 possible resistor combinations in total. In order to maintain a constant bandwidth $R_{ideal,j}$, j=1,2,…,125, follows a logarithmic curve if j defines an ordering such that $R_{ideal,j}+1 > R_{ideal,j}$. The problem was to find the optimum selection of 12 resistor values that constitute a best-fit approximation of $R_{ideal,j}$. The resistor values had to be taken from a fixed set of discrete resistor values of the E-192 series. Eventually the problem was tackled with DE using the variant DE/rand/1/exp and NP=50. As recommended in [47] the conductance values were kept non-discretized during difference vector generation. Discretization of the values was deferred to the evaluation of the cost function value according to (19). It took only 500 generations and roughly 3 minutes on an Intel E6750@2.66GHz processor to obtain a satisfactory solution shown in Fig. below. The resulting frequency error of the PLL stayed well within +/- 0.5MHz for all input frequencies from 25MHz to 340MHz

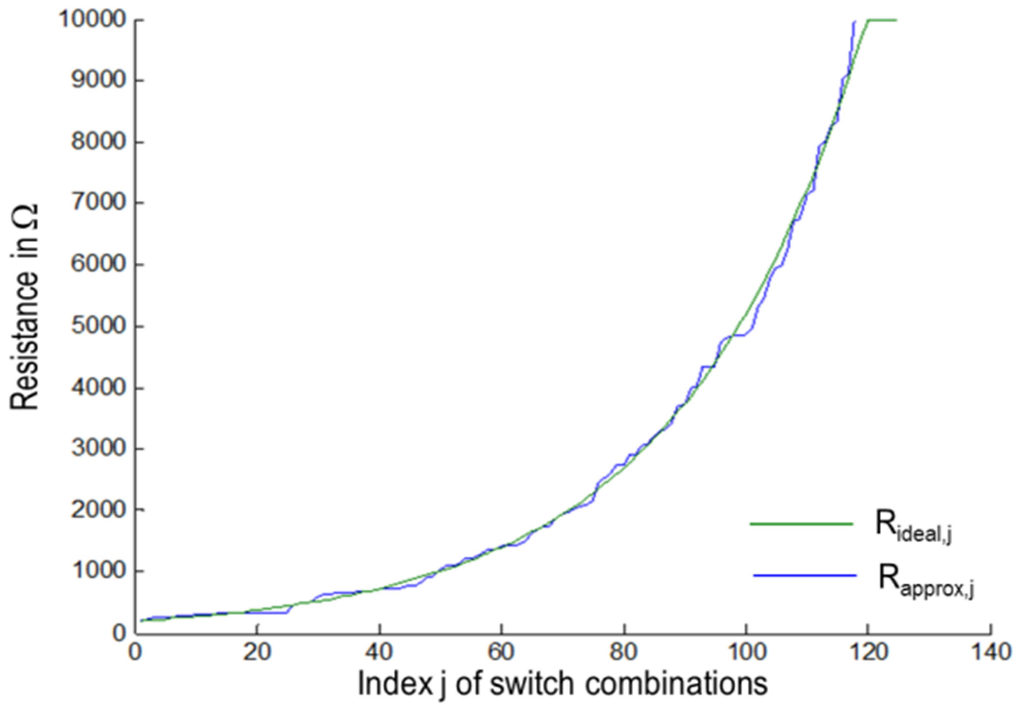


*Fig. 2.7: Final approximation of $R_{ideal,j}$ with $R_{approx,j} = 1/G_j$.*

### *2.5.4 SQUELCH OPTIMIZATION:*

The squelch function is an important element in airborne radios like the MR6000L/R ® made by Rohde & Schwarz. It suppresses the audio output of the radio receiver when the desired signal does not have sufficient S/N (signal-to-noise) ratio and/or signal strength in order to not disturb the pilot with irritating noise or crackling sounds. The squelch characteristic has usually evolved through many iterations stemming from customer feedback as well as extensive lab and field tests under various noise conditions, so that the user finally experiences the most convenient squelch behavior. In order to free up DSP cycles to allow the implementation of additional features into the MR6000L/R the computational effort for the squelch function had to be reduced. The task here was to ensure that the carefully obtained squelch characteristic remained the same. Fig. 8 shows the squelch of the MR6000L/R which consists of an S/N squelch and a so-called carrier squelch. Both of them are illustrated for the F3E-mode [47]. In this application it is desired to reduce processor cycles in the S/N squelch because the carrier squelch did not offer sufficient potential for DSP cycle reduction. We considered the following situation: the squelch operates in F3E mode on signals with a sample rate of f1 = 128 kHz. Consequently, all IIR filters (HP,LP) also have a clock rate of 128 kHz. According to the Nyquist Theorem [43] an audio signal of 3.4kHz bandwidth can be sampled with much lower frequency than 128 kHz, so in order to reduce the computational complexity, we wanted to reduce the sampling frequency of the LP filter to 32 kHz, i.e., only every fourth sample was to be considered by the LP filter. The straightforward approach to go further would have been to redesign both the HP and LP filter by applying standard filter design theory and then tune the remaining parameters of the S/N squelch using the radio and the existing test & measurement setups including customer feedback. In order to avoid the latter we decided on a different approach: a MATLAB ® model of the entire S/N squelch was established which had to be verified first by applying audio files to both the radio and the model and then comparing the results. After the model had been verified the plan was to optimize all parameters simultaneously such that the squelch behavior in the model would remain the same after the sample rate reduction from 128kHz to 32kHz.

## 2.6 Conclusions:

This chapter introduced one of the most useful optimization techniques Differential Evolution with its various steps & pseudo codes. The real world applications of differential evolution are also exhibited in this chapter. Apart from the mentioned applications DE can be used in function optimization like Single Objective Optimization, Multiobjective Optimization. Multiobjective Optimization (MO) problems consist of several competing and incommensurable objective functions. Such problems are frequently encountered in numerous scientific and engineering applications. The need for the concurrent minimization of more than one objective functions, renders the use of EAs particularly attractive. In contrast to traditional gradient–based techniques, EAs operate on a set of potential solutions of the problem. Thus, EAs are capable of detecting several solutions of an MO problem in a single run.

## *References*

[1] Kenneth V. Price ,Rainer M. Storn Jouni A. Lampinen, 'Differential Evolution- A Practical Approach to Global Optimization', Springer-Verlag Berlin Heidelberg 2005.

[2] Swagatam Das, P.N.Suganthan, 'Differential Evolution: A Survey of the State-of-the-Art', IEEE Transaction on Evolutionary Computation, Vol. 15, No. 1, February 2011.

[3] S. Das, A.Abraham, U. K. Chakraborty, and A. Konar "Differential
evolution using a neighborhood based mutation operator," IEEE Trans.
Evol. Comput., vol. 13, no. 3, pp. 526–553, Jun. 2009.

[4] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition based differential evolution," IEEE Trans. Evol. Comput., vol. 12, no. 1, pp. 64–79, Feb. 2008

[5] J. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," IEEE Trans. Evol. Comput., vol. 13, no. 5, pp. 945–958, Oct. 2009

[6] J. Brest, S. Greiner, B. Boskovi ˇ c, M. Mernik, and V. ´ Zumer, "Self- adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., vol. 10, no. 6, pp. 646–657, Dec. 2006

[7] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," IEEE Trans. Evol. Comput., vol. 13, no. 2, pp. 398–417, Apr. 2009

[8] F. Neri and V. Tirronen, "Recent advances in differential evolution: A. review and experimental analysis," Artif. Intell. Rev., vol. 33, no. 1, pp. 61–106, Feb. 2010

[9] S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," in Proc. IEEE Congr. Evol. Comput., vol. 1. Sep. 2005, pp. 443–450.

[10] L. V. Santana-Quintero and C. A. Coello Coello, "An algorithm based on differential evolution for multiobjective problems," Int. J. Comput. Intell. Res., vol. 1, no. 2, pp. 151–169, 2005.

[11] F. Xue, A. C. Sanderson, and R. J. Graves, "Pareto-based multiobjective differential evolution," in Proc. Congr. Evol. Comput., vol. 2. 2003, pp. 862–869.

[12] T. Robic and B. Filipic, “DEMO: Differential evolution for multiobjective optimization,” in Proc. 3rd Int. Conf. Evol. Multi-Criterion Optimization, LNCS 3410. 2005, pp. 520–533.
[13] A. W. Iorio and X. Li, “Solving rotated multiobjective optimization problems using differential evolution,” in Proc. AI: Adv. Artif. Intell.,
LNCS 3339. 2004, pp. 861–872.
[14] B. V. Babu and M. M. L. Jehan, “Differential evolution for multiobjective optimization,” in Proc. Congr. Evol. Comput., vol. 4. Dec. 2003, pp. 2696–2703.
[15] H. Li and Q. Zhang, “Multiobjective optimization problems with
complicated Pareto sets, MOEA/D and NSGA-II,” IEEE Trans. Evol.
Comput., vol. 13, no. 2, pp. 284–302, Apr. 2009.
[16] B. Y. Qu and P. N. Suganthan, “Multiobjective evolutionary algorithms based on the summation of normalized objectives and diversified selection,” Inform. Sci., vol. 180, no. 17, pp. 3170–3181, Sep. 2010.
[17] K. Price, R. Storn, and J. Lampinen, Differential Evolution—A Practical Approach to Global Optimization. Berlin, Germany: Springer, 2005.
[18] R. Storn, “On the usage of differential evolution for function optimization,” in Proc. North Am. Fuzzy Inform. Process. Soc., 1996, pp. 519–523.
[19] R. Storn, “Differential evolution design of an IIR-filter with requirements for magnitude and group delay,” in Proc. IEEE Int. Conf. Evol. Comput., 1996, pp. 268–273.
[20] J. Lampinen and I. Zelinka, “Mixed integer-discrete-continuous optimization with differential evolution,” in Proc. 5th Int. Mendel Conf.
Soft Comput., Jun. 1999, pp. 71–76.
[21] R. Storn, “System design by constraint adaptation and differential
evolution,” IEEE Trans. Evol. Comput., vol. 3, no. 1, pp. 22–34, Apr.
1999.
[22] M. Zhang, W. Luo, and X. Wang, “Differential evolution with dynamic stochastic selection for constrained optimization,” Inform. Sci., vol. 178, no. 15, pp. 3043–3074, Aug. 2008. 40
[23] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello, and K. Deb, “Problem definitions and evaluation criteria for the CEC 2006 (special session on constrained real-parameter optimization),” Nanyang Technol. Univ., Singapore,

Tech. Rep., 2006.
[24] V. L. Huang, A. K. Qin, and P. N. Suganthan, “Self-adaptive differential evolution algorithm for constrained real-parameter optimization,” in Proc. IEEE Congr. Evol. Comput., Jul. 2006, pp. 324–331.
[25] N. Noman and H. Iba, “Enhancing differential evolution performance with local search for high dimensional function optimization,” in Proc. Conf. Genet. Evol. Comput., Jun. 2005, pp. 967–974.
[26] Y. Gao and Y. Wang, “A memetic differential evolutionary algorithm for high dimensional function spaces optimization,” in Proc. 3rd ICNC 20, vol. 4. Aug. 2007, pp. 188–192.
[27] Z. Yang, K. Tang, and X. Yao, “Differential evolution for highdimensional function optimization,” in Proc. IEEE Congr. Evol. Comput., Sep. 2007, pp. 3523–3530.
[28] Z. Yang, J. He, and X. Yao, “Making a difference to differential evolution,” in Advances in Metaheuristics for Hard Optimization, Z. Michalewicz and P. Siarry, Eds. Berlin, Germany: Springer, 2007, pp.415–432.
[29] Z. Yang, K. Tang, and X. Yao, “Large scale evolutionary optimization using cooperative coevolution,” Inform. Sci., vol. 178, no. 15, pp. 2985– 2999, 2008.
[30] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen, and Z. Yang, “Benchmark functions for the CEC’2008 special session and competition on large scale global optimization,” Nature Inspired Comput. Applicat. Lab., USTC, China, Nanyang Technol. Univ., Singapore, Tech. Rep., 2007.
[31] T. Krink, B. Filipiˇc, and G. B. Fogel, “Noisy optimization problems: A particular challenge for differential evolution,” in Proc. IEEE Congr. Evol. Comput., 2004, pp. 332–339.
[32] S. Das, A. Konar, and U. Chakraborty, “Improved differential evolution algorithms for handling noisy optimization problems,” in Proc. IEEE Congr. Evol. Comput., vol. 2. 2005, pp. 1691–1698.
[33] B. Liu, X. Zhang, and H. Ma, “Hybrid differential evolution for noisy optimization,” in Proc. IEEE Congr. Evol. Comput., Jun. 2008, pp. 587–592.
[34] R. Mendes and A. S. Mohais, “DynDE: A differential evolution for dynamic optimization problems,” in Proc. IEEE Congr. Evol. Comput., vol. 2. 2005, pp. 2808–2815.

[35] R. Thomsen, “Multimodal optimization using crowding-based differential evolution,” in Proc. IEEE Congr. Evol. Comput., 2004, pp. 1382– 1389.
[36] B. Y. Qu and P. N. Suganthan, “Novel multimodal problems and differential evolution with ensemble of restricted tournament selection,” in Proc. IEEE Congr. Evol. Comput., Jul. 2010, pp. 3480–3486.
[37] Storn, R., “Differential Evolution – Ein praktischer Ansatz zur globalen Parameteroptimierung, Presentation at the public seminar “Elektronische Bauelemente” at Technical University of Munich, May 17, 2004.
[38] ITU-T Recommendation G.992.1, Asymmetric digital subscriber line  (ADSL) transceivers, June 1999.
[39] Infineon AG, GEMINAX ADSL Transceiver Chipset PEB55508 (GEMINAX-D), PEB 22720 (GEMINAX-A0), and PEB 22716 (GEMINAX-L2)
[40] A. Van Dyck, “Preferred Numbers”, Proceedings of the Institute of Radio Engineers, volume 24, pp. 159-179, Feb. 1936.
[41] K. Borst, “Entwicklung eines universellen, FPGA-basierten Simulators für Mobilfunkkanäle”, Diploma Thesis, Univ. Stuttgart, Dez. 2005.
[42] Storn, R., “Optimization of Wireless Communication Applications using Differential Evolution”, SDR Technical Conference SDR’07, Denver, Colorado, Nov. 5-9, 2007
[43] L. R. Rabiner & B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, 1975
[44] MATLAB, https://www.mathworks.com/
[45] R. Storn, “Designing Nonstandard Filters with Differential Evolution”, IEEE Signal Processing Magazine, pp. 103 – 106, 2005.
[46] J. Schöller, “Bestimmung der PLL Schleifenverstärkung”, Rohde&Schwarz-internal communication, July 22, 2011.
[47] Determination of Necessary Bandwidths Including Examples for their Calculation, Recommendation ITU-R SM.1138, Geneva, 1995.

# *CHAPTER-3*

## Bacterial Foraging Optimization Algorithm

### 3.1 Introduction:-

Bacterial foraging optimization algorithm (BFOA) becomes popular as a global optimization algorithm of current interest for distributed optimization and control. BFOA is inspired by the social foraging behaviour of Escherichia coli & M.xanthus. BFOA has already drawn the attention of researchers because of its efficiency in solving real-world optimization problems arising in several application domains. The underlying biology behind the foraging strategy of E.coli is emulated in an extraordinary manner and used as a simple optimization
Algorithm. Bacterial Foraging Optimization Algorithm (BFOA), was proposed by Passino [1], is a new comer to the family of nature-inspired optimization algorithms. During last five decades, optimization algorithms like Genetic Algorithms (GA) [2], Evolutionary Programming (EP) [3], Evolutionary Strategies (ES) [4], have been dominating the realm of optimization algorithms. Recently natural swarm inspired algorithms like Particle Swarm Optimization (PSO) [5], Ant Colony Optimization (ACO) [6] have found their way into this domain and proved their effectiveness. Following the same trend of swarm-based algorithms, Passino proposed the BFOA in [1]. Application of group foraging strategy of a swarm of E.coli bacterial in multi-optimal function optimization is the key idea of the new algorithm. Bacterial search for nutrients in a manner to maximize energy obtained per unit time. Individual bacterium also communicates with others by sending signals. A bacterium takes foraging decisions after considering two previous factors. The process, in which a bacterium moves by taking small steps while searching for nutrients, is called chemotaxis and key idea of BFOA is mimicking chemotactic movement of virtual bacterial in the problem search space. Since its inception, BFOA has drawn the attention of researchers from diverse fields of knowledge especially due to its biological motivation and graceful structure. Researchers are trying to hybridize BFOA with different other algorithms in order to explore its local and global search properties separately. It has already been applied to many real world problems and proved its effectiveness over many variants of GA and PSO.

Mathematical modelling, adaptation, and modification of the algorithm might be a major part of the research on BFOA in future.

### 3.1.1 Bacterial Foraging: E. coli:

A typical E. coli bacterium[1] is shown in Figure. It has a plasma membrane, cell wall, and capsule that contain, for instance, the cytoplasm and nucleoid. The pili (singular, pilus) are used for a type of gene transfer to other E. coli bacteria, and flagella (singular, flagellum) are used for locomotion. (Only one is shown, but in the actual cell there are as many as six.) The cell is about 1μm in diameter, and 2μm in length. The E. coli cell only weighs about 1 picogram, and is composed of about 70% water. Salmonella typhimurium is a similar type of bacterium. When E. coli grows, it gets longer, then divides in the middle into two,"daughters." Given sufficient food and held at the temperature of the human gut (one place where they live) of 37 deg.C . E. coli can synthesize and replicate everything it needs to make a copy of itself in about 20 min.; hence, growth of a population of bacteria is exponential with a relatively short "time to double" the population size.

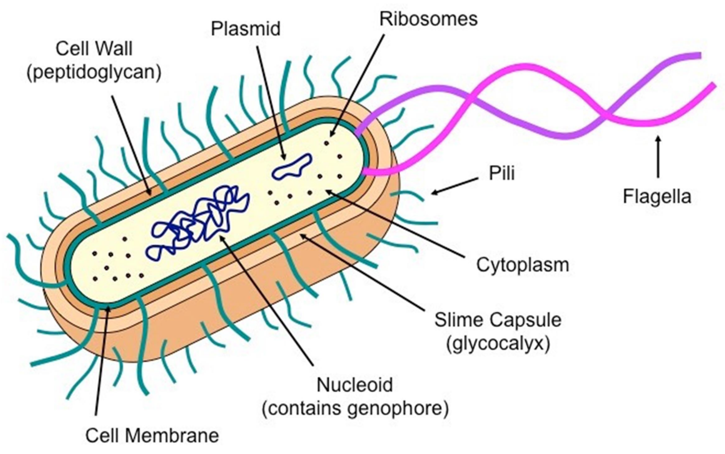

Cell Wall (peptidoglycan)
Plasmid
Ribosomes
Pili
Flagella
Cytoplasm
Slime Capsule (glycocalyx)
Nucleoid (contains genophore)
Cell Membrane

*Fig.3.1: Different Parts of an Escherichia coli*

E. coli bacterium has a control system that enables it to search for food and try to avoid noxious substances (the resulting motions are called "taxes" ). For instance, it swims away from alkaline and acidic environments, and towards more neutral ones. To explain the motile behaviour of E. coli bacteria,its actuator (the flagella), "decision-making," sensors, and closed-loop behavior (i.e., how it moves in various environments–its "motile behavior") will be explained.

#### 3.1.1.1 Swimming and Tumbling via Flagella:

Locomotion is achieved via a set of relatively rigid flagella that enable it to "swim" via each of them rotating in the same direction at about 100 – 200 revolutions per second (in control systems terms, we think of the flagella as providing for actuation). Each flagellum is a left-handed helix configured so that as the base of the flagellum (i.e., where it is connected to the cell) rotates counter clockwise, as viewed from the free end of the flagellum looking towards the cell, it produces a force against the bacterium so it pushes the cell. If a flagellum rotates clockwise, then it will pull at the cell. From an engineering perspective, the rotating shaft at the base of the flagellum is quite an interesting contraption that seems to use what biologists call a "universal joint" (so the rigid flagellum can, "point",in different directions, relative to the cell). In addition, the mechanism that creates the rotational forces to spin the flagellum in either direction is described by biologists as being a biological "motor" (a relatively rare contraption in biology even though several types of bacteria use it) as shown in Figure 3.2

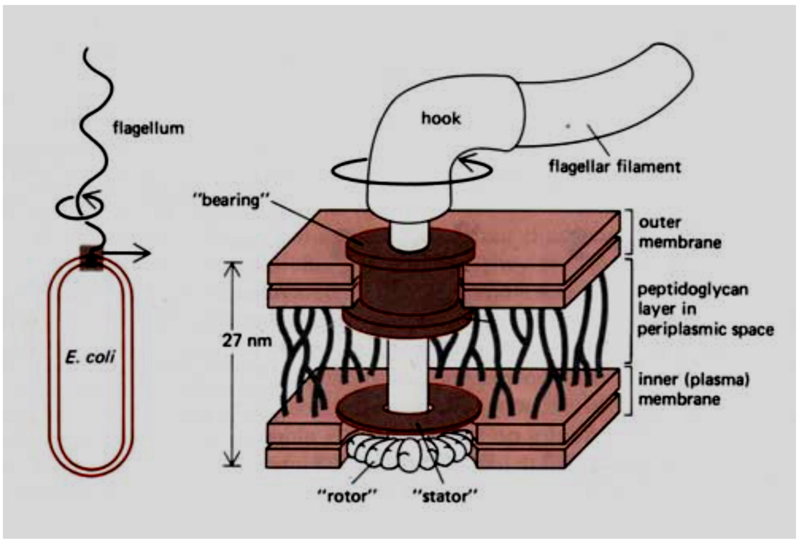


*Fig. 3.2: Magnified view of flagellum-cell body joint*

An E. coli bacterium can move in two different ways: it can "run" (swim for a period of time) or it can “tumble,” and it alternates between these two modes of operation its entire lifetime (i.e., it is rare that the flagella will stop rotating). If the flagella rotate clockwise, each flagellum pulls on the cell and the net effect is that each flagellum operates relatively independent of the others and so the bacterium “tumbles” about (i.e., the bacterium does not have a set direction of movement and there is little displacement). To tumble after a run, the cell slows down or stops first; since bacteria are so small they experience almost no inertia, only viscosity, so that when a bacterium stops swimming, it stops within the diameter of a proton. Call the time interval during which a tumble occurs a “tumble interval.”

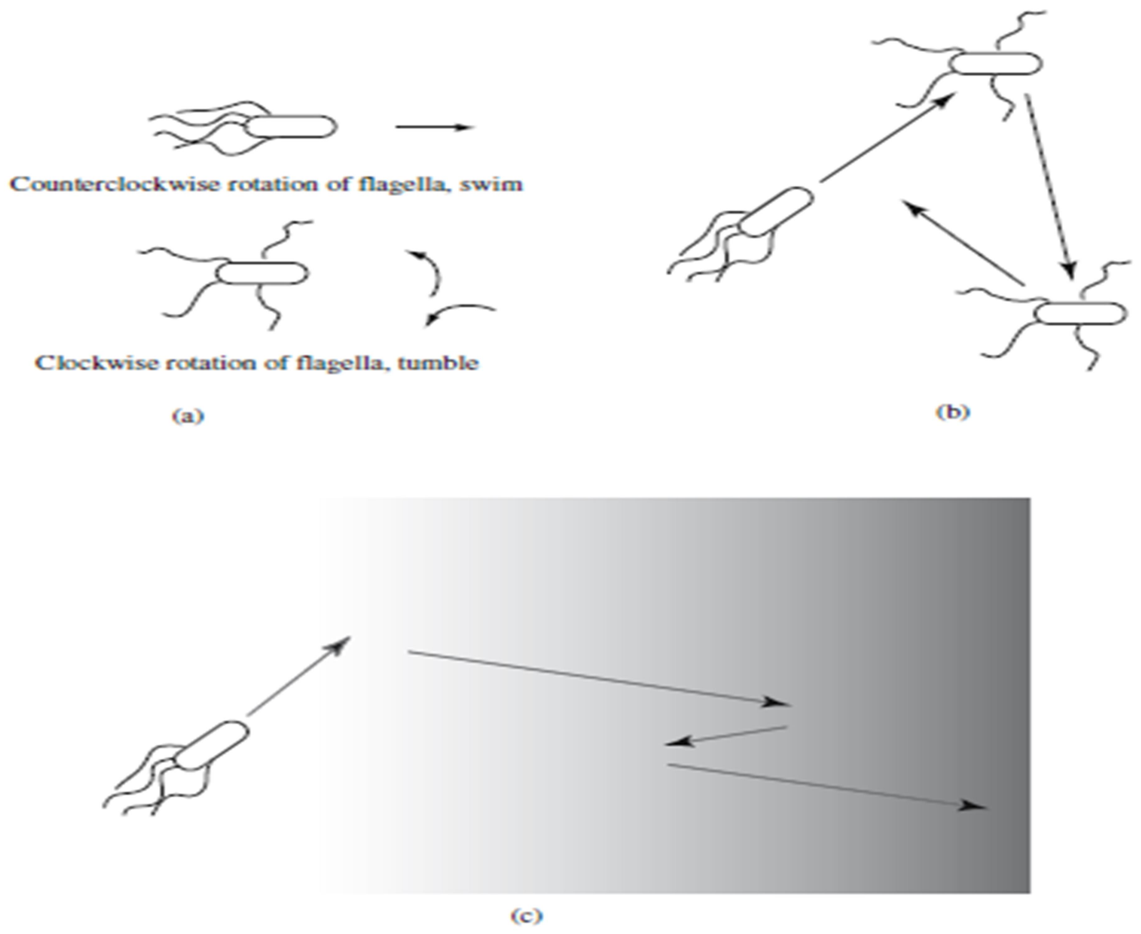

Counterclockwise rotation of flagella, swim
Clockwise rotation of flagella, tumble
(a)
(b)
(c)

*Fig. 3.3: Bundling phenomenon of flagella shown in (a), swimming and tumbling behavior of the E. coli bacterium is shown in (b) in a neutral medium and in (c) where there is a nutrient concentration gradient, with darker shades indicating higher concentrations of the nutrient.*

If the flagella move counter clockwise, their effects accumulate by forming a “bundle” (it is thought that the bundle is formed due to the viscous drag of the medium) and hence, they essentially make a “composite propeller” and push the bacterium so that it runs (swims) in one direction.(Fig.3.3)

### 3.1.1.2 Bacterial Motile Behavior: Climbing Nutrient Gradients:-

The motion patterns (called “taxes”) that the bacteria will generate in the presence of chemical attractants and repellents are called “chemotaxes.” For E. coli, encounters with serine or aspartate result in attractant responses, while repellent responses result from the metal ions Ni and Co, changes in pH, amino acids like leucine, and organic acids like acetate. What is the resulting emergent pattern of behavior for a whole group of E. coli bacteria? Generally, as a group they will try to find food and avoid harmful phenomena, and when viewed under a microscope, we will get a sense that a type of intelligent behavior has emerged, since they will seem to intentionally move as a group (analogous to how a swarm of bees moves).

To explain how chemotaxis motions are generated, we simply must explain how the E. coli decides how long to run since, from the above discussion, we know what happens during a tumble or run. First, note that if an E. coli is in some substance that is neutral, in the sense that it does not have food or noxious substances, and if it is in this medium for a long period of time (e.g., more than one minute), then the flagella will simultaneously alternate between moving clockwise and counterclockwise so that the bacterium will alternately tumble and run. This alternation between the two modes will move the bacterium, but in random directions, and this enables it to “search” for nutrients. For instance, in the isotropic homogeneous environment described above, the bacteria alternately tumble and run with the mean tumble and run lengths given above, and at the speed that was given. If the bacteria are placed in a homogeneous concentration of serine (i.e., one with a nutrient but no gradients), then a variety of changes occur in the characteristics of their motile behavior. For instance, mean run length and mean speed increase and mean tumble time decreases. They do, however, still produce a basic type of searching behavior; even though it has some food, it persistently searches for more. Next, suppose that the bacterium happens to encounter a nutrient gradient .The change in the concentration of the nutrient triggers a reaction such that the bacterium will spend more time swimming and less time tumbling. As long as it travels on a positive concentration gradient (i.e., so that it moves towards increasing nutrient concentrations) it will tend to lengthen the time it spends swimming (i.e., it runs farther). The directions of movement are "biased" towards increasing nutrient gradients. The cell does not change its direction on a run due to changes in the gradient—the tumbles basically determine the direction of the run, aside from the Brownian influences mentioned above suppose that the concentration of the nutrient is constant for the region it is in, after it has been on a positive gradient for some time. In this case, after a period of time (not immediately), the bacterium will return to the same proportion of swimming and tumbling as when it was in the neutral substance so that it returns to its standard searching behavior. It is never satisfied with the amount of surrounding food; it always seeks higher concentrations. Actually, under certain experimental conditions, the cell will compare the concentration observed over the past 1 sec. with the concentration observed over the 3 sec. before that and it responds to the difference . Hence, it uses the past 4 sec. of nutrient concentration data to decide how long to run . Considering the deviations in direction due to Brownian movement discussed above, the bacterium basically uses as much time as it can in making decisions about climbing gradients. In effect, the run length results from how much climbing it has done recently. If it has made lots of progress and hence, has just had a long run, then even if for a little while it is observing a homogeneous medium (without gradients), it will take a longer run. After a certain time period, it will recover and return to its standard behavior in a homogeneous medium. Basically, the bacterium is trying to swim from places with low concentrations of nutrients to places with high concentrations. An opposite type of behavior is used when it encounters noxious substances. If the various concentrations move with time, then the bacteria will try to "chase" after the more favorable environments and run from harmful ones. Clearly, nutrient and noxious substance diffusion and motion will affect the motion patterns of a group of bacteria in complex ways.

### 3.1.1.3 Underlying Sensing and Decision-Making Mechanisms:-

A cross-section of one corner of the E. coli bacterium is shown in the figure. The sensors are the receptor proteins, which are signaled directly by external substances or via the "periplasmic substrate-binding proteins." The "sensor" is very sensitive, in some cases requiring less than 10 molecules of attractant to trigger a reaction, and attractants can trigger a swimming reaction in less than 200 ms.

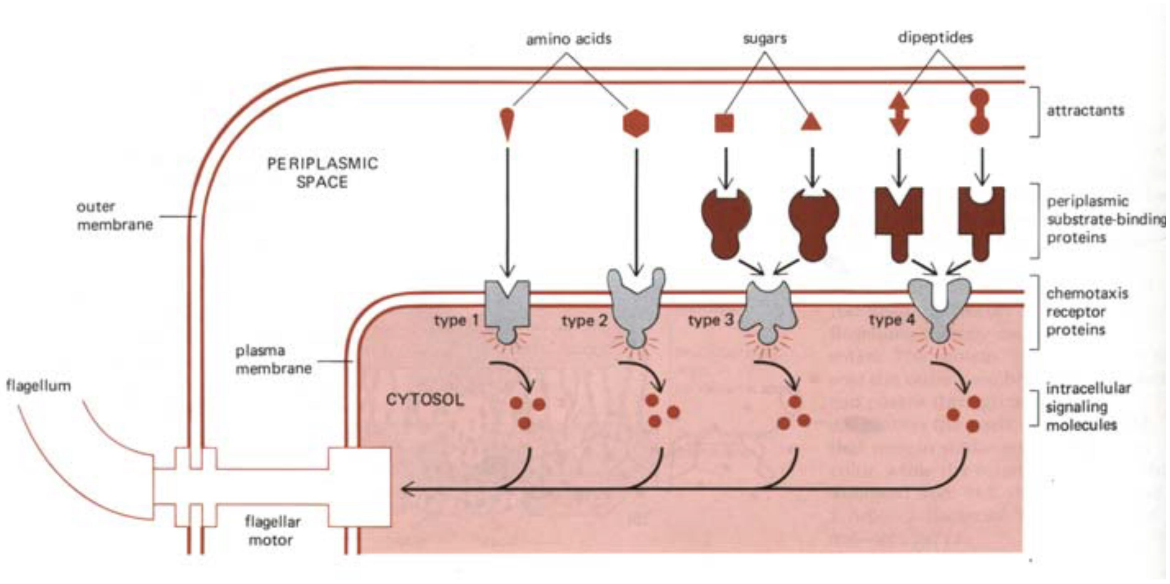


*Fig.3.4: E. coli bacterium, flagellar connection, and biological "motor"*

Although at first glance it seems possible that the bacterium senses concentrations at both ends of the cell and finds a simple difference to recognize a concentration gradient (a spatial derivative) but this is not the case. Experiments have shown that it performs a type of sampling, and roughly speaking, it remembers the concentration a moment ago, compares it with a current one, and makes decisions based on the difference like Euler approximation. So, it can be concluded that with memory, a type of addition mechanism, an ability to make comparisons, a few simple internal "control rules," and its chemical sensing and locomotion capabilities, the bacterium is able to achieve a complex type of searching and avoidance behaviour. Evolution has designed this control system. It is robust and clearly very successful at meeting its goals of survival when viewed from a population perspective.

### 3.1.1.4 **Elimination and Dispersal Events**:-

It may happen that the local environment where a population of bacteria lives changes either gradually (e.g., via consumption of nutrients) or suddenly due to some other influence. There can be events such that all the bacteria in a region are killed or a group is dispersed into a new part of the environment. For example, local significant increases in heat can kill a population of bacteria that are currently in a region with a high concentration of nutrients (one can think of heat as a type of noxious influence). Or, it may be that water or some animal will move populations of bacteria from one place to another in the environment.Over long periods of time, such events have spread various types of bacteria into virtually every part of our environment, from our intestines, to hot springs and underground environments, and so on. The effect of elimination and dispersal events on chemotaxis is destroy of chemotactic progress but it also has the effect of assisting in chemotaxis since dispersal may place bacteria near good food sources. From a broad perspective, elimination and dispersal is part of the population-level motile behavior.

### **3.1.1.5 Evolution of Bacteria:-**

Mutations in E. coli occur at a rate of about 10−7 per gene, per generation. In addition to mutations that affect its physiological aspects (e.g., reproductive efficiency at different temperatures), E. coli bacteria occasionally engage in a type of “sex” called “conjugation,” where small gene sequences are unidirectionally transferred from one bacterium to another. It seems that these gene sequences apparently carry good fitness characteristics in terms of reproductive capability, so conjugation is sometimes thought of as a transmittal of “fertility.” To achieve conjugation, a pilus extends to make contact with another bacterium, and the gene sequence transfers through the pilus. While conjugation apparently spreads “good” gene sequences, the “homogenizing effect” on gene frequency from conjugation is relatively small compared to how sex works in other organisms. This is partly since conjugation is relatively rare, and partly since the rate of reproduction is relatively high, on the order of hours depending on environmental conditions. Due to these characteristics, population genetics for E. coli may be dominated by selection sweeps triggered by the acquisition, via sex, of an adaptive allele.

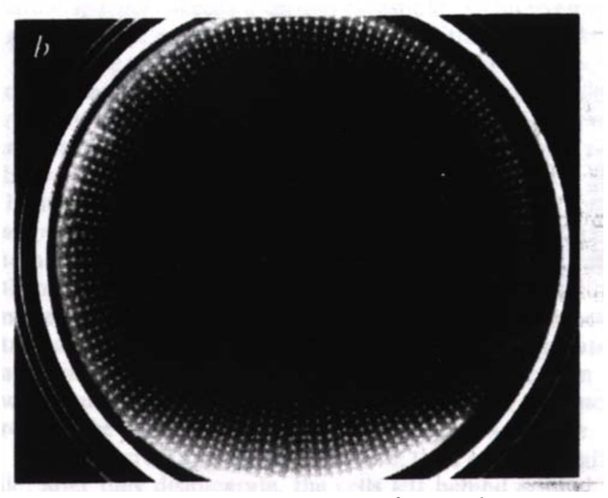


*Fig.3.5:Swarm Pattern of E. Coli*

## 3.2 Literature review:

In last two decades, multiple nature-inspired optimization algorithms have been proposed, including Genetic Algorithm (GA) [2], Particle Swarm Optimization (PSO) [5], and Ant Colony Optimization (ACO) [6]. Based on the competitive-cooperative mechanism of Escherichia

coli.(E. coli) in the foraging process, Passino [1] proposed a novel swarm intelligence algorithm called Bacterial Foraging Optimization algorithm (BFO), which consists mainly of four behaviors: chemotaxis, swarming, reproduction and elimination-dispersal. Two important classes of population-based optimization algorithms are evolutionary algorithms [3] and swarm intelligence-based algorithms [7]. Swarm Intelligence is a meta-heuristic method in the field of artificial intelligence that is used to solve optimization problems. It is based on the collective behaviour of social insects, flocks of birds, or schools of fish. A swarm can be considered as any collection of interacting agents or individuals. Researchers have analysed such behaviours and designed algorithms that can be used to solve combinatorial and numerical optimization problems in many science and engineering domains. Recent studies[8],[9] have shown that algorithms based on Swarm Intelligence have great potential. Bacterial Foraging Optimization Algorithm (BFOA), is a new algorithm in the category of swarm intelligence. The group foraging behaviour of E.coli bacteria in multi-optimal function optimization is the basis of the new algorithm. Bacteria search for nutrients in a manner to maximize energy obtained per unit time. The communication of Individual bacteria is done by sending various signals. The foraging decisions of a bacteria are based on two factors one is called chemotaxis and another is mimicking chemotactic movement. The process, in which a bacterium moves by taking small steps while searching for nutrients, is called chemotaxis and key idea of BFOA is mimicking chemotactic movement of virtual bacteria in the problem search space. After invention of the BFOA, it has been applied to various Engineering and Science related optimization problems. Researchers are continuously modifying the BFOA by hybridization, inserting new phase and new control parameters to improve the performance. This algorithm has been already compared with GA, PSO and ABC[10] algorithms for solving real world optimization problems. Bacteria are in the category of social insects. The foraging behaviour of bacteria produces an intelligent social behaviour, called as swarm intelligence. This swarm intelligence is simulated and an intelligent search algorithm namely, Bacterial Foraging Optimization (BFO) algorithm. Since its inception, a lot of research has been carried out to make BFO more and more efficient and to apply BFO for different types of problems. In order to get rid of the drawbacks of basic BFO, researchers have improved BFO in many ways.

In his paper Qian Zhang[11] proposed algorithm Chaotic BFO combines two chaotic strategies. First, a chaotic initialization strategy is incorporated into BFO for bacterial population initialization. Then, a chaotic local search with a `shrinking' strategy is introduced into the chemotaxis step. This proposed `shrinking' strategy is a modified version of the method described in[12].

***Chaos Theory:-*** Over the last few decades, much progress has been made in the chaos theory. It has been used widely in different fields of science such as chaos control[13], feature selection, and parameter optimization . Chaotic sequences have three basic dynamic properties: sensitive dependence on initial conditions, randomicity, and ergodicity. Chaotic sequences have been applied to various metaheuristic optimization algorithms in recent years. In [14], a novel GA with chaotic mutation was proposed by replacing the Gaussian mutation operator in real-coded GA with a chaotic mapping. Mingjun and Huanwen [15] introduced chaotic initialization and chaotic sequences into Simulated Annealing (SA) instead of Gaussian distribution. In [16]. In order to improve the overall searching performance of basic algorithms, other metaheuristic optimization algorithms also use the chaos theory, including Moth-Flame Optimization (MFO) [17], Firefly Algorithm (FA) [18], Artificial Bee Colony (ABC) [10], Biogeography- Based Optimization (BBO) [19], Krill Herd (KH) [20],Water Cycle Algorithm(WCA) [21], and Grey Wolf Optimizer (GWO) [22]. If chaos variables are used in the search, more advantage is gained over random search. The basic idea of chaos optimization is 1) to introduce chaos state into optimization variables by using a similar carrier method, 2) to magnify the traversal range of chaotic motions to the range of optimization variables, and 3) to use the chaos variables to search to make the search more effective. The proposed method generates a chaotic sequence using logistic mapping as shown in equation below:

$$ch_{i+1} = \mu\ ch_{i+1} * (1-ch_i) \quad i=1,\ldots S-1 \quad \ldots\ldots(1)$$

Where μ is the control parameter, take μ=4. 0 < ch1 < 1, and ch1 ≠ 0:25; 0:5; 0:75; 1. Not difficult to prove that when μ =4, the system is completely in chaos. S is the number of individuals.

Chaotic search usually works well in local optimization for its ergodicity and randomicity [23], [24]. However, its performance decreases when it explores a large search space. To overcome this shortcoming, chaotic local search was introduced. Due to the randomicity of chaotic local search, the search process can avoid premature convergence and local optima stagnation. In [25], chaotic local search was incorporated into PSO to construct a chaotic PSO (CPSO), where the parallel population-based evolutionary searching ability of PSO and chaotic searching behavior are reasonably combined. Jia *et al.* [26] proposed an effective memetic DE algorithm called DECLS, which utilizes a chaotic local search with a `shrinking' strategy. In [27], a chaotic local search was integrated into the reduced Symbiotic Organisms Search (SOS) to form chaotic SOS (CSOS) for improving solution accuracy and convergence mobility.

### *a) Chaotic Initialization:-*

Step 1: Through the chaos mechanism, the chaotic sequence is generated by using the logistic map generated by the Eq. (1),

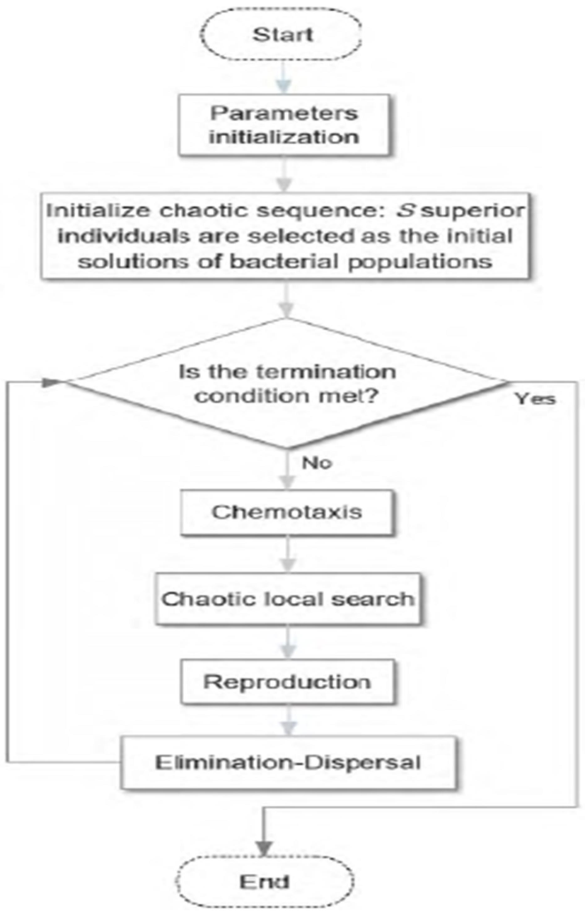

Start
Parameters initialization
Initialize chaotic sequence: S superior individuals are selected as the initial solutions of bacterial populations
Is the termination condition met?
Yes
No
Chemotaxis
Chaotic local search
Reproduction
Elimination-Dispersal
End

*Fig.3.6: Overall procedure of Chaotic BFO.*

The position P of the initial bacterial population is mapped into the chaotic sequence to generate the position PCh of the corresponding chaotic initial bacterial population. As shown in Eq. (2):

$$PCh = ch_i * P \quad i = 1,\ldots,S \ \ \ldots..(2)$$

Step 2: From the initial position P of the bacterial population and its corresponding position PCh of the chaotic initial bacterial population, S superior individuals are selected as the initial solutions of bacterial populations. Loop execution (S - 1) times.

***b) Chaotic Initialization:-***

*Step 1:* Before the chemotaxis operation of the *i*th bacterium, place the *i*th bacterium at the position of *j*th chemotaxis, *k*th reproductive, and *l*th elimination dispersal as the optimal positio*ng best* in the current bacterial population.

*Step 2:* The chaotic variable *chi* generated in Eq. (1) is mapped into the chaotic vector *CHi* in the domain of definition [*lb*, *ub*], as shown in Eq. (3):

$$CHi = lb + chi *( ub - lb) \quad i = 1,\ldots\ldots, S \quad \ldots\ldots.(3)$$

Where lb and ub represent the lower and upper bounds of the initial solution, respectively.

*Step 3:* The chaotic vector CHi is linearly combined with the optimal position *gbest* to generate the candidate bacterial position sol, as shown in Eq. (4):

$$sol =(1 - setCan)*gbest + setCan* CHi \quad i= 1,\ldots\ldots.., S \quad \ldots\ldots\ldots(4)$$

Where setCan is the contraction factor, which is determined by Eq. (5):

$$setCan = exp(-Intertime/max\_iteration) \quad \ldots\ldots\ldots.(5)$$

Where Max_iteration represents the maximum number of iterations of the algorithm and Intertime represents the current iteration number of the algorithm. From Eq. (5), it can be seen that the contraction factor setCan decreases as the number of iterations increases. As shown in Eq. (4), the smaller the value of setCan is, the smaller the range of chaos search is. In the early iteration, setCan is larger, which helps to expand the search range and increase the diversity of the population. At the later stage of iteration , setCan is smaller, which helps to converge to the global optimal solution.

*Step 4:* If the candidate bacterial position sol is better than Gbest (Gbest represents the current optimal fitness function of the bacterial population), the fitness of sol is recorded as Gbest and gbest (gbest described in Step 1 represents the optimal position in the current bacterial population) is updated as sol. If the chaotic sequence length reaches S, local search ends; otherwise skip to Step 2 to continue execution. BFOA was implemented in image segmentation by N.Sanyal et al.[28]. Entropy based thresholding results in more centralized distribution of image histogram of segmented images. This work is related to thresholding based image segmentation in image processing domain. Bi-level thresholding is performed for some electronically available web based benchmark images of size 512X512. Optimal thresholds of image histograms are found out by maximizing the Fuzzy Entropy. Optimization tool is developed on the basis of Bacterial Foraging Algorithm. Most of the evolutionary algorithms deal with minimization problems. Bacterial foraging is also suitable for solving minimization problems. On the contrary entropy based thresholding is a maximization problem. Fuzzy entropy itself is a negative function (Zhao et al., 2001). Therefore in the proposed algorithm, minimization of negative fitness function eventually results in maximization of absolute value of fuzzy entropy. In 8 bit representation, any gray image can be represented by 256 gray scale intensity levels. Uniformity is a quantitative measure by which the quality of segmented image can be evaluated. Uniformity is defined as –

$$u = 1\text{-}2*c* \frac{\sum_{j=0}^{p} \sum_{j \in Rj} (fi - \mu j)2}{N*(fmax - fmin)2}$$

where,
c = Number of thresholds
Rj = jth segmented region
N = Total number of pixels in given image
fi =  Gray level of pixel i
lj = Mean gray level of pixels of jth region
fmax = Maximum gray level of pixels of jth region
fmin = Minimum gray level of pixels of jth region

Uniformity is a positive fraction between 0 and 1. The higher the value of uniformity, better is the quality of the segmented image. Since the proposed problem is a stochastic optimization problem for a new application area, there are  too many free parameters which can affect the system performance. It is  basically a challenge for a researcher to find the best set of free parameters for which the system performance is satisfactory. For this purpose classical BFOA has undergone twenty independent runs for a particular parameter.  setting (each of the parameters are varied one at a time and its effect on the performance was noted) and considering the best performance parameter set is recorded. During this run two benchmark images 'lena' and 'pepper' were chosen and the manner in which the fitness function and the uniformity vary are recorded. in below tables. At first, size of the population of bacteria colony is varied in steps from 20 to 60. Order of the population is arbitrarily chosen by surveying the literature (Maitra & Chatterjee, 2008). With the increase in number of bacteria the optimization algorithm must yield better result because at least one bacterium is expected to be nearest to the optimum point (Passino, 2002).

Table 1 shows that the mean entropy

| | Mean uniformity | Mean entropy | Maximum uniformity |
|---|---|---|---|
| Lena | | | |
| ***S=20*** | 0.947 | 9.9664 | 0.9576 |
| ***S=30*** | 0.9478 | 9.9667 | 0.9576 |
| ***S=40*** | 0.9471 | 9.9854 | 0.9578 |
| ***S=50*** | 0.9461 | 9.9974 | 0.9565 |
| ***S=60*** | 0.9452 | 9.995 | 0.9577 |
| Pepper | | | |
| ***S=20*** | 0.9521 | 10.0888 | 0.9654 |
| ***S=30*** | 0.9486 | 10.1133 | 0.9653 |
| ***S=40*** | 0.951 | 10.127 | 0.9652 |
| ***S=50*** | 0.9474 | 10.1273 | 0.964 |
| ***S=60*** | 0.9488 | 10.1271 | 0.9634 |

*Table-3.1 Impact of variation of number of bacteria in BFOA when applied for segmentation of images lena and pepper.*

| | Mean uniformity | Mean entropy | Maximum uniformity |
|---|---|---|---|
| Lena | | | |
| ***Nc=10*** | 0.9469 | 9.9847 | 0.9564 |
| ***Nc=15*** | 0.9471 | 9.9854 | 0.9578 |
| ***Nc=20*** | 0.9439 | 9.9909 | 0.9537 |
| ***Nc=25*** | 0.946 | 9.9904 | 0.9576 |
| Pepper | | | |
| ***Nc=10*** | 0.9514 | 10.1187 | 0.9649 |
| ***Nc=15*** | 0.951 | 10.127 | 0.9652 |
| ***Nc=20*** | 0.949 | 10.1097 | 0.9652 |
| ***Nc=25*** | 0.9514 | 10.1332 | 0.9641 |

Table-3.2 Impact of variation of number of chemotactic steps in BFOA when applied for segmentation of images lena and pepper.

| | Mean uniformity | Mean entropy | Maximum uniformity |
|---|---|---|---|
| Lena | | | |
| ***Ns=5*** | 0.9474 | 9.9762 | 0.9576 |
| ***NS=10*** | 0.9471 | 9.9854 | 0.9578 |
| ***Ns=15*** | 0.9448 | 9.9861 | 0.9559 |
| ***NS=20*** | 0.9454 | 9.9899 | 0.9561 |
| Pepper | | | |
| ***Ns=5*** | 0.9496 | 10.1166 | 0.9653 |
| ***NS=10*** | 0.951 | 10.127 | 0.9652 |
| ***Ns=15*** | 0.9483 | 10.1222 | 0.9648 |
| ***NS=20*** | 0.9472 | 10.1046 | 0.9639 |

Table-3.3 Impact of variation of swim lengths in BFOA when applied for segmentation of images lena and pepper.

| | Unit run length | Mean uniformity | Mean entropy | Maximum uniformity |
|---|---|---|---|---|
| ***Lena*** | 0.01 | 0.947 | 9.9818 | 0.9577 |
| | 0.1 | 0.9448 | 9.9855 | 0.9559 |
| | 0.4 | 0.9471 | 9.9854 | 0.9578 |
| | 1.0 | 0.944 | 10.0322 | 0.9545 |
| ***Pepper*** | 0.01 | 0.9492 | 10.0901 | 0.9652 |
| | 0.1 | 0.9461 | 10.101 | 0.9653 |
| | 0.4 | 0.951 | 10.127 | 0.9652 |
| | 1.0 | 0.9473 | 10.1755 | 0.9622 |

Table-3.4 Impact of variation of Run-Length Unit in BFOA when applied for segmentation of images lena and pepper.

| Parameter | Value |
|---|---|
| S | 40 |
| Nc | 15 |
| Ns | 10 |
| C(i,k) | 0.4 |
| Nre | 1 |
| Ned | 2 |
| Ped | 0.4 |

Table-3.5 Free parameters for BFOA.

| | Mean uniformity | Mean entropy | Maximum uniformity |
|---|---|---|---|
| Lena | | | |
| ***Ɛ=1*** | 0.9476 | 10.0018 | 0.9576 |
| ***Ɛ=2*** | 0.9465 | 10.0009 | 0.9577 |
| ***Ɛ=3*** | 0.9468 | 9.9917 | 0.9559 |
| ***Ɛ=4*** | 0.944 | 10.0077 | 0.955 |
| | | | |
| Pepper | | | |
| ***Ɛ=1*** | 0.9495 | 10.1466 | 0.9646 |
| ***Ɛ=2*** | 0.9473 | 10.1243 | 0.964 |
| ***Ɛ=3*** | 0.9479 | 10.1242 | 0.9639 |
| ***Ɛ=4*** | 0.9554 | 10.1273 | 0.9651 |

Table-3.6 Impact of variation precision goal of bacteria in ABFOA when applied for segmentation of images lena and pepper.

Table 1 shows that the mean entropy, i.e. the fitness function of the algorithm, maximizes with the increase in number of bacteria. We choose S = 40 as an increase in the number of bacteria results in increase in computational complexity and uniformity measure will also increase in most occasions. Once S is fixed, we started varying number of chemotactic steps Nc from 10 to 25. Though the computational complexity increases with the increase in number of chemotactic steps, the chance of getting optimum result also increases .For both mean uniformity, and maximum uniformity Nc = 15 gives considerably good result for both the images. Hence Nc is fixed at 15. Now as Nc is fixed we can vary Ns which physically signifies that random walk is biased more in the direction of climbing down the hill. For both the test images Ns = 10 delivers the almost best set of results. So Ns is fixed at 10.

## 3.3 Main steps of BFOA:-

***Chemotaxis*** **:** This process simulates the movement of an E.coli cell through swimming and tumbling via flagella. Biologically an E.coli bacterium can move in two different ways. It can swim for a period of time in the same direction or it may tumble, and alternate between these two modes of operation for the entire lifetime. Suppose ( j, k, l) i q represents i-th bacterium at jth chemotactic, k-th reproductive and l-th elimination-dispersal step. C(i) is the size of the step taken in the random direction specified by the tumble (run length unit). Then in computational chemotaxis the movement of the bacterium may be represented by

$$\theta^{i}(j+1,k,l) = \theta^{i}(j,k,l)+C(i)\frac{\Delta(i)}{\sqrt{\Delta T(i)\Delta(i)}} \quad \ldots\ldots(1)$$

Where $\Delta$ indicates a vector in the random direction whose elements lie in [-1, 1].

***Swarming:*** An interesting group behavior has been observed for several motile species of bacteria including E.coli and S. typhimurium, where intricate and stable spatio-temporal patterns (swarms) are formed in semisolid nutrient medium. A group of E.coli cells arrange themselves in a traveling ring by moving up the nutrient gradient when placed amidst a semisolid matrix with a single nutrient chemo-effecter. The cells when stimulated by a high level of succinate, release an attractant aspertate, which helps them to aggregate into groups and thus move as concentric patterns of swarms with high bacterial density. The cell-to-cell signaling in E. coli swarm may be represented by the following function.

$$Jcc(\theta,P(j,k,l)) = \sum_{i=1}^{s} Jcc\ (\theta, \theta i(j,k,l))$$
$$= \sum_{i=1}^{s}[-d\ attractant\ exp(-Wattarctant \sum_{m=1}^{p}(\theta_m - \theta_m^i)2)]+\sum_{i=}^{s}[h_{repellant}\ exp(-w_{repellant} \sum_{m=1}^{p}(\theta_m - \theta_m^i)2)] \quad \ldots\ldots(2)$$

where $J_{cc}(\theta , P( j, k, l))$ cc q is the objective function value to be added to the actual objective function (to be minimized) to present a time varying objective function, S is the total number of bacteria, p is the number of variables to be optimized, which are present in each bacterium and $\theta = [\theta_1, \theta_2, \dots \theta_p]$T is a point in the p-dimensional search domain. $d_{atractant}$ ,$w_{attractant}$ ,$d_{repellant}$ ,$w_{repellant}$ are different coefficients that should be chosen properly.

***Reproduction:*** The least healthy bacteria eventually die while each of the healthier bacteria (those yielding lower value of the objective function) asexually split into two bacteria, which are then placed in the same location. This keeps the swarm size constant.

***Elimination and Dispersal:*** Gradual or sudden changes in the local environment where a bacterium population lives may occur due to various reasons e.g. a significant local rise of temperature may kill a group of bacteria that are currently in a region with a high concentration of nutrient gradients. Events can take place in such a fashion that all the bacteria in a region are killed or a group is dispersed into a new location. To simulate this phenomenon in BFOA some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space.

## 3.4 PSUDO Code of BFOA:-

***Parameters:***

```
Step1- Initialize parameters p, S, $N_c$, $N_s$, $N_{re}$, $N_{ed}$, $P_{ed}$, C(i)(i=1,2…S), $\theta^i$
```

***Algorithm:***

```
Step-2 Elimination-dispersal loop: l=l+1
Step-3 Reproduction loop: k=k+1
Step-4 Chemotaxis loop: j=j+1
[a] For i =1,2…S take a chemotactic step for bacterium i as follows.
[b] Compute fitness function, J (i, j, k, l).
    Let, $J(i, j, k, l) = J(i, j, k, l) + J_{cc}(\theta_i(j, k, l), P(j, k, l))$ (add on the cell-to cell attractant–repellant profile to simulate the swarming behavior)
where, Jcc is defined in (2).

[c] Let $J_{last}$=J (i, j, k, l) to save this value since we may find a better cost via a run.
```

[d] Tumble: generate a random vector D(i)ÎÂp with each element (i),m 1,2,..., p, m D = a random number on [-1, 1].

[e] Move: Let $\theta^{i}$ *(j+1,k,l)* = $\theta^{i}$ *(j,k,l)+C(i)*$\frac{\Delta(i)}{\sqrt{\Delta T(i)\Delta(i)}}$

This results in a step of size C(i) in the direction of the tumble for bacterium i.

[f] Compute J (i, j +1, k, l) and let

*J(i,j+1,k,l)=j(i,j,k,l)+* $J_{cc}$ *θi (j+1,k,l), P( j+1, k, l)*

[g] Swim

i) Let m=0 (counter for swim length).

ii) While m< s N (if have not climbed down too long).

Let m=m+1.

If J (i, j +1, k, l) < $J_{last}$ ( if doing better), let Jlast = J (i, j +1, k, l) and

Let $\theta^{i}$ *(j+1,k,l)* = $\theta^{i}$ *(j,k,l)+C(i)*$\frac{\Delta(i)}{\sqrt{\Delta T(i)\Delta(i)}}$

And use this ( j 1, j, k) i q + to compute the new J (i, j +1, k, l) as we did in [f]

Else, let m= s N . This is the end of the while statement.

[h] Go to next bacterium (i+1) if i ¹ S (i.e., go to [b] to process the next bacterium).

Step-5 If c j < N , go to step 4. In this case continue chemotaxis since the life of the bacteria is not over.

Step-6 Reproduction:

[a] For the given k and l, and for each i = 1,2,..., S , let

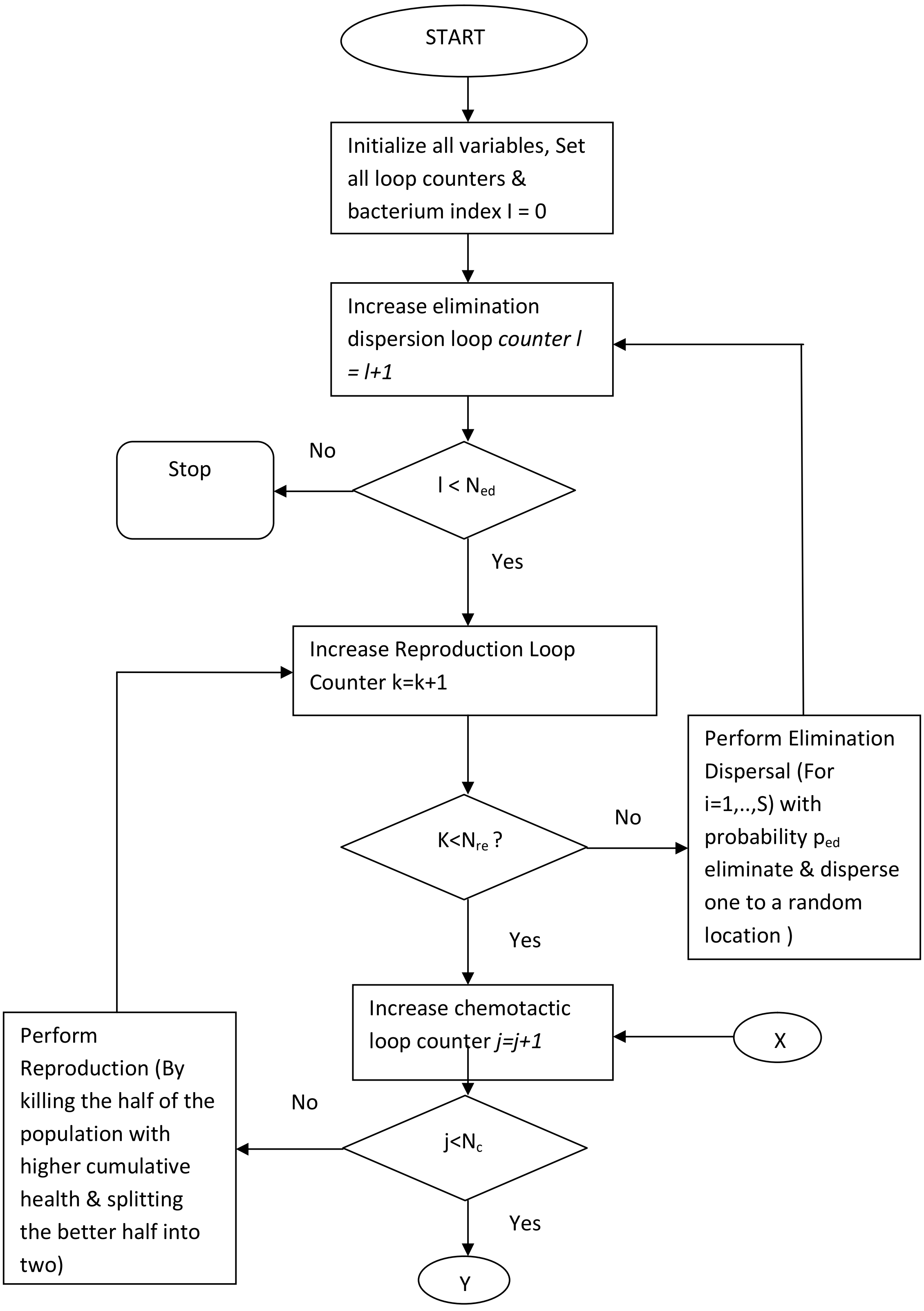
$$J^{i}_{healt} = \sum_{j=1}^{Nc+1} J(i,j,k,l) \quad \ldots\ldots(3)$$

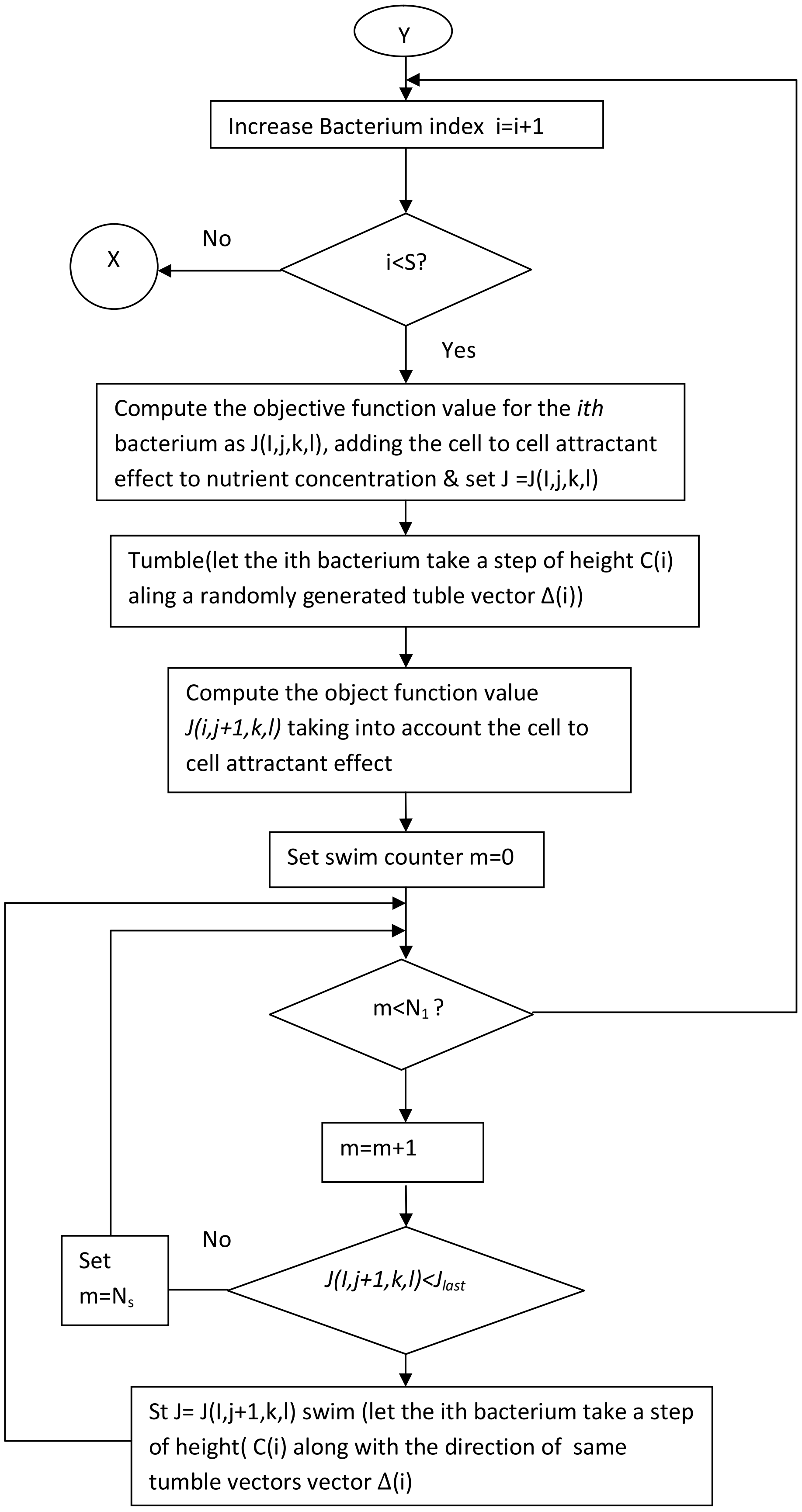
be the health of the bacterium i (a measure of how many nutrients it got over its lifetime and how successful it was at avoiding noxious substances). Sort bacteria and chemotactic parameters *C(i)* in order of ascending cost health $J_{health}$ (higher cost means lower health).

[b] The r S bacteria with the highest health J values die and the remaining r S bacteria with the best values split (this process is performed by the copies that are made are placed at the same location as their parent).

Step-7 If $k < N_{re}$ , go to step 3. In this case, we have not reached the number of specified reproduction steps, so we start the next generation of the chemotactic loop.

Step-8 Elimination-dispersal: For i = 1,2..., S with probability ed P , eliminate and disperse each bacterium (this keeps the number of bacteria in the population constant). To do this, if a bacterium is eliminated, simply disperse another one to a random location on the optimization domain. If ed $l < N$ , then go to step 2; otherwise end.


START
Initialize all variables, Set all loop counters & bacterium index I = 0
Increase elimination dispersion loop *counter l = l+1*
$l < N_{ed}$
No
Stop
Yes
Increase Reproduction Loop Counter k=k+1
$K<N_{re}$ ?
No
Perform Elimination Dispersal (For i=1,..,S) with probability $p_{ed}$ eliminate & disperse one to a random location )
Yes
Increase chemotactic loop counter *j=j+1*
X
$j<N_c$
No
Perform Reproduction (By killing the half of the population with higher cumulative health & splitting the better half into two)
Yes
Y



Y
Increase Bacterium index i=i+1
No
X
i<S?
Yes
Compute the objective function value for the *ith* bacterium as J(I,j,k,l), adding the cell to cell attractant effect to nutrient concentration & set J =J(I,j,k,l)
Tumble(let the ith bacterium take a step of height C(i) aling a randomly generated tuble vector Δ(i))
Compute the object function value *J(i,j+1,k,l)* taking into account the cell to cell attractant effect
Set swim counter m=0
m<N1 ?
m=m+1
No
Set m=Ns
*J(I,j+1,k,l)<Jlast*
St J= J(I,j+1,k,l) swim (let the ith bacterium take a step of height( C(i) along with the direction of same tumble vectors vector Δ(i)

Flowchart of BFOA

## 3.5 Real world applications:-

### *3.5.1 BFOA Based Adaptive PID controller:*

The PID controller has been widely used in the most industrial process due to simple structure, algorithm, and good performance. However, the PID controller parameters are still computed using the classic tuning formulae and these do not provide good control performance in all situations, for example, for unstable systems with time delay. In order to provide consistent, reliable, safe and optimal solution to industrial control problems as described above, many approaches for PID control schemes and tuning techniques have been presented. These schemes generally consist of four basic parts: model estimation, desired system specifications, optimal tuning mechanism and an online PID.

PID controller is a simple algorithm based on estimating the information of the past, present and future. The control system mainly combined with PID controller and charged objects. The PID transfer function is:

$$G_p = K_p + K_i / S + K_d$$

The performance of a PID controller can be enhanced upto a greater extent by using BFOA[29]. A basic system with BFOA is shown in figure below:-

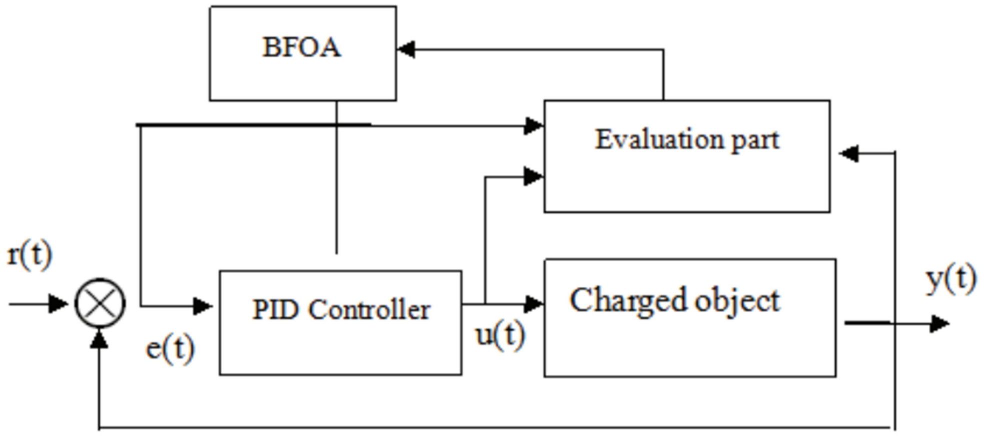


*Fig.3.7: BFOA Based Adaptive PID controller block diagram*

Where r (t) is as the system input, and e (t) is as the system error, u (t) is as
the PID controller output, y (t) is as the system output. Evaluation part, that is the objective function, means the index of the fitness value which is calculated by using of input variables, output variables and intermediate variables. Although different control systems have different objective fitness function and even the control systems have changed dramatically, the algorithm has little change relatively. Even the transfer function has been changed, three parameters $K_p$ , $K_i$, $K_d$ can be obtained easily, this is the fundamental principle of adaptive PID controller which can be realized . Analog PID which discrete into digital PID results can be expressed simply as:

$$u(k) = k_p e(k) + k_i \sum_{j=0}^{k} e(j)T + k_d (e(k)-e(k-1))/T$$

Where $k_p$, $k_i$, $k_d$ are the coefficient of Proportional-Integral-Derivative (PID). T is the sampling time. K is the consequence of sampling. e(k) is the error of the point of k times. The BFOA encoding usually use binary code or real number code, the real number code is proposed according to the above equation. Then the parameters $k_p$,$k_i$,$k_d$ and T are initialized.

### *3.5.2 BFOA Harmonic reduction in Inverter:*

Multilevel inverter is attaining increasing attention in the past few years because of
its high voltage and therefore high power capability [1, 2]. There are various modulation methods which includes sinusoidal PWM (SPWM) and space-vector modulation (SVM) to control the output voltage and to reduce unwanted components in multilevel inverters. Selective Harmonic Elimination (SHE) is a technique choosing the switching times so that specific lower order dominant harmonics such as 5th, 7th, and 11th and so on are suppressed in the output voltage connected to the load. The SHE technique involves solving the nonlinear transcendental equations characterizing the harmonic contents, which offers multiple solutions. These equations can be solved through optimization techniques effectively. There are several optimization algorithms such as GA (Genetic algorithm) which emulates the process of biological evolution, PSO (Particle Swarm Optimization) inspired by social behavior of bird flocking, ACO (Ant Colony Optimization) emulating the foraging behavior of ant colonies, Artificial Bee Colony Algorithm (ABC) mimicking foraging behavior of swarm of honey bees have been used

extensively for the solving the non-linear equations. Here, *BFOA* is realized to minimize lower-order harmonics, and to maintain the desired fundamental component. Among the different topology structures of MLI, Cascaded H-bridge topology has gained more prominence due to their simplicity of its structure and control, modularity and flexibility. The cascaded MLI consists of a series connected H bridge inverter units. Each of the individual cell (full bridge unit) can generate three different output voltage levels: +Vdc, 0, and −Vdc. All the H Bridges connected in series can produce staircase waveform.

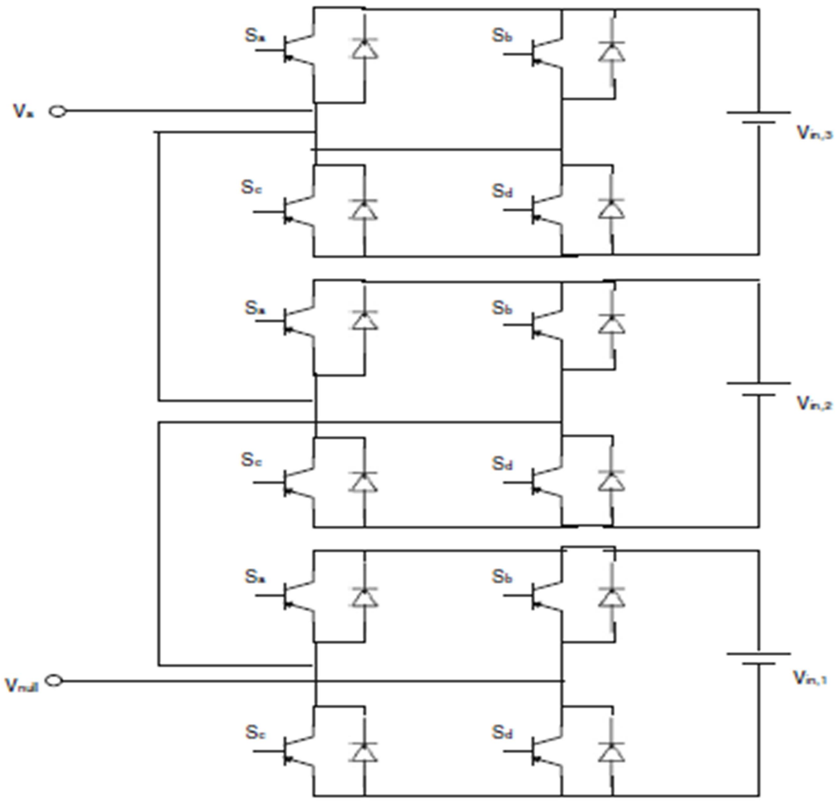


*Fig.3.8: level cascaded multilevel inverter*

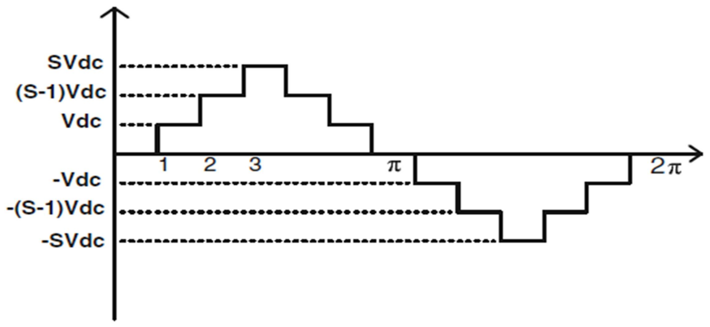


*Fig.3.9: Output voltage waveform of a 7-level MLI*

where Vn is the nth harmonic component amplitude. The angles of the switches should be restricted between 0 and π/2 ($0 \le \theta < \pi/2$). Because of odd quarter-wave symmetry, even order harmonics become zero.

The objective of SHEPWM is to eliminate the lower order dominant harmonics. In a 7-level Cascaded MLI the 5th and 7th dominant lower order harmonics are to be eliminated because 3rd harmonic will be eliminated in 3 phase systems. So, in order to obtain the desired fundamental harmonic and to eliminate 5th and 7th harmonics, three non-linear equations with three switching angles are shown in below set of equations:-

$$\left.\begin{aligned} V_1 &= \frac{4V_{dc}}{\pi}\,[\cos\theta_1 + \cos\theta_2 + \cos\theta_3] \\ V_5 &= \frac{4V_{dc}}{5\pi}\,[\cos5\theta_1 + \cos5\theta_2 + \cos5\theta_3] \\ V_7 &= \frac{4V_{dc}}{7\pi}\,[\cos7\theta_1 + \cos7\theta_2 + \cos7\theta_3] \end{aligned}\right] \text{.........(1)}$$

In above set of equations $V_5$ and $V_7$ are set to zero in order to eliminate 5th and 7th harmonics, respectively. In order to obtain various switching angles a titled modulation index, new index, is defined to be a representative as:

$$MI \triangleq \frac{V1}{\frac{12V_{dc}}{\Pi}} \; (0 \le MI \le 1)\text{.........(2)}$$

Here, MI is between 0 and 1 in order to cover various values of V1. Thus, by replacing equation (2) in equation (1)

$$MI = \frac{1}{3}[\cos\theta_1 + \cos\theta_2 + \cos\theta_3]$$

$$0 = [\cos5\theta_1 + \cos5\theta_2 + \cos5\theta_3]$$

$$0 = [\cos7\theta_1 + \cos7\theta_2 + \cos7\theta_3]$$

Now, the three different switching angles, namely $\theta_1$, $\theta_2$, and $\theta_3$, can be found with respect to the range of MI. The parameters to be initialized are number of bacteria required for searching which are initialized as 26 and number of iterations to be undertaken, taken here as 100. The fitness function that is used is given below-

$$f = 100\left[\frac{V_1^* - V_1}{V_1^*}\right]^4 + \sum_{S=2}^{S}\frac{1}{h_s}\left[50\frac{V_{h_s}}{V_1}\right]^2$$

| | |
|---|---|
| $V_1^*$ | Desired fundamental voltage |
| $V_1$ | Fundamental voltage |
| $h_s$ | Order of sth harmonic |
| $V_{h_s}$ | Voltage of sth harmonics |

The switching angles thus found eliminate the low-order harmonics (5th and 7th) while retaining the maximum fundamental component.

### *3.5.3Application of BFOA in Economic Load Dispatch :*

In the current scenario, where the industries are plagued with global energy crisis and skyrocketing fuel prices, the need of the hour is efficient utilization of the available resources without compromising the demand. In order to solve the complexity involved in ED problems, a power system has to be operated in such a way that it eventually supplies all the loads with minimum cost. According to [30] the classic mode of solving ED issues was through performing lambda-iteration and gradient methods, which requires the unit input-output curves of generators. However, due to forbidden operating zones these curves do not show a monotonic rise. Thus, as per [31], for optimizing non-linear cost functions, the traditional ED algorithms are not generally recommended. The economic load dispatch is an important aspect of modern Power systems among others like unit commitment, load forecasting, available transfer capacity calculation, security analysis and scheduling of fuel purchases etc. The power system ELD problem based on BFOA and FA has been tested on 3-generator and 13 -generator system. The initial system for testing comprises of three generating units with a demand of 850MW and 900MW for both inclusion and exclusion of valve point loading. The second test system consists of 13 generating units with a total demand of 1800Mw with both including and excluding of valve point loading. From the paper of [32]it can be said that the test results of BFOA outweigh FA in accuracy. Its peers are computationally intensive and time taking because these are based on stochastic (i.e. behaviorally non deterministic) searches in population and generations. But BFOA method is self-adaptive. It converges in a better fashion and hence gives better results with regards to generation cost. Thus it takes lesser memory space to converge to the optimal value. The established respect of bacterial movement for food search lends further credence to this method's usage. Also it avoids premature convergence and the BFOA's second stage i.e reproduction process paces the convergence process.

## 3.6 Conclusions:-

BFOA is very much suitable to achieve global optimum. This is because of the inherent characteristics of the BFOA itself comprising of elimination and dispersal processes. Some real world practical applications of Bacterial Foraging Algorithm is described here. Apart from these example the BFOA can be used in different domains like, uninhabited autonomous vehicles (UAVs) that are used in military (or commercial) applications where i) Animals, organisms = UAVs , ii) social foragers = group of cooperating UAVs that can communicate with each other, iii) prey, nutrients = targets, iv) predators, noxious substances = threats, and v) environment = battlefield. it would be interesting to characterize the physiological and environmental aspects that drove evolution to "design" a specific foraging strategy and optimize its operation. This would help us understand how vehicular constraints and tactical situations affect the design and operation of the cooperative Strategy. Any superior performance in one class of problems generally results in inferior performance over another class. The combination of such algorithms is called hybridization or hybrid metaheuristics. They combine the advantages of individual algorithms while overcoming individual weaknesses. Such an approach ensures that at least one of the algorithms provides optimum solution to each particular class of problems. The main concern while using hybridization is that the percentage of successful convergence to global optimum should increase as opposed to those obtained by standalone algorithm.

## *References:*

[1] Kevin M. Passino,”Biomimicry for Optimization, Control, and Automation”, Springer-Verlag London Limited 2005 .

[2] John H. Holland,’ Adaptation in Natural and Artificial Systems’, 1992 The MIT Press.

[3] David J. Fogel, Lawrence J. Fogel,’ An introduction to evolutionary programming’, Natural selection Inc. 3333North Torrey Pines Ct. Suite 200

[4] Ingo Rechenberg, Hans-Paul Schwefel, Hans-Michael Voigt, Werner Ebeling, Parallel Problem Solving from Nature- PPSN IV, International Conference on Evolutionary Computation - The 4th International Conference on Parallel Problem Solving from Nature Berlin, Germany,September 22-26, 1996.

[5] James Kennedy, Russell C. Eberhart, Yuhui Shi , Swarm Intelligence’, The Morgan Kaufmann Series in Evolutionary Computation.

[6] Marco Dorigo, Thomas Stützle, ‘Ant Colony Optimization’, 2004 Massachusetts Institute of Technology.

[7] Gerardo Beni , Jing Wang,’ Swarm Intelligence in Cellular Robotic Systems’, College of Engineering University of California, Riverside, Springer-Verlag Berlin Heidelberg 1993.

[8] Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of
IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948.
IEEE (1995).

[9] Price, K.V., Storn, R.M., Lampinen, J.: Differential evolution: a practical approach to global optimization. Springer, Heidelberg (2005) .

[10] Derviş Karaboga, Bahriye Akay, Artificial Bee Colony (ABC) Algorithm on Training Artificial Neural Networks, Bilgisayar Mühendisliği Bölümü Erciyes Üniversitesi, Kayseri.

[11] Qian Zhang, Huiling Chen , Jie Luo, Yueting Xu, Chengwen Wu, and Chengye Li,Chaos
Enhanced Bacterial Foraging Optimization for Global Optimization’’, 10.1109/ACCESS.
2018.2876996

[12] D. Jia, G. Zheng, and M. K. Khan, ``An effective memetic differential evolution algorithm based on chaotic local search," Inf. Sci., vol. 181, no. 15, pp. 3175_3187, 2011.

[13] L. Zhang and C. Zhang, ``Hopf bifurcation analysis of some hyperchaotic systems with time-delay controllers," Kybernetika, vol. 44, no. 1, pp. 35_42, 2008.

[14] L. J. Yang and T. L. Chen, ``Application of chaos in genetic algorithms," Commun. Theor. Phys., vol. 38, no. 2, pp. 168_172, 2002.

[15] J. Mingjun and T. Huanwen, ``Application of chaos in simulated annealing," Chaos, Solitons Fractals, vol. 21, no. 4, pp. 933_941, 2004.

[16] B. Alatas, E. Akin, and A. B. Ozer, ``Chaos embedded particle swarm optimization algorithms," Chaos, Solitons Fractals, vol. 40, no. 4, pp. 1715_1734, 2009

[17] M.Wang et al., ``Toward an optimal kernel extreme learning machine using a chaotic moth-Flame optimization strategy with applications in medical diagnoses," Neurocomputing, vol. 267, pp. 69_84, Dec. 2017.

[18] A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi, ``Firefly algorithm with chaos," Commun. Nonlinear Sci. Numer. Simul., vol. 18, no. 1, pp. 89_98, 2013.

[19] S. Saremi, S. Mirjalili, and A. Lewis, ``Biogeography-based optimization with chaos," Neural Comput. Appl., vol. 25, no. 5, pp. 1077_1097, 2014.

[20] G.-G. Wang, L. H. Guo, A. H. Gandomi, G.-S. Hao, and H. Q. Wang,Chaotic krill herd algorithm," Inf. Sci., vol. 274, pp. 17_34, Aug. 2014.

[21] A. A. Heidari, R. Ali Abbaspour, and A. R. Jordehi, ``An ef_cient chaotic water cycle algorithm for optimization tasks," Neural Comput. Appl., vol. 28, no. 1, pp. 57_85, 2017.

[22] M. Kohli and S. Arora, ``Chaotic grey wolf optimization algorithm for
constrained optimization problems," J. Comput. Des. Eng., vol. 5, no. 4, pp. 458_472, Oct. 2018.

[23] M. S. Tavazoei and M. Haeri, ``Comparison of different one-dimensional
maps as chaotic search pattern in chaos optimization algorithms," Appl. Math. Comput., vol. 187, no. 2, pp. 1076_1085, 2007.

[24] L. Wang, D.-Z. Zheng, and Q. S. Lin, ``Survey on chaotic optimization methods," Comput. Technol. Autom., vol. 20, no. 1, pp. 1_5, 2001.

[25] B. Liu, L. Wang, Y.-H. Jin, F. Tang, and D. X. Huang, ``Improved particle swarm optimization combined with chaos," Chaos, Solitons Fractals, vol. 25, no. 5, pp. 1261_1271, 2005.

[26] D. Jia, G. Zheng, and M. K. Khan, ``An effective memetic differential evolution algorithm based on chaotic local search," Inf. Sci., vol. 181, no. 15, pp. 3175_3187, 2011.

[27] S. Saha and V. Mukherjee, ``A novel chaos-integrated symbiotic organisms search algorithm for global optimization," Soft Comput., vol. 22, no. 11, pp. 3797_3816, Jun. 2018.

[28] Nandita Sanyal, Amitava Chatterjee, Sugata Munshi, An adaptive bacterial foraging algorithm for fuzzy entropy based image segmentation'', 2011 Elsevier Ltd.

[29] Guozhong Wu,"Application of Adaptive PID Controller Based On Bacterial Foraging Optimization Algorithm".

[30] Sinha, N.; Chakrabarti, R.; Chattopadhyay, P.K., "Evolutionary programming techniques for economic load dispatch,", IEEE Transactions on Evolutionary Computation , vol.7, no.l, pp. 83-94, Feb 2003

[31] W. J. Tang, M. S. Li, Q. H. Wu and J. R. Saunders, "Bacterial Foraging Algorithm for Optimal Power Flow in Dynamic Environments", IEEE Transactions on Circuits and Systems-I, vo1.55, no.8, pp.2433-2442, Sept. 2008.

[32] D.R.Prabha, A.K. Prasad. Raju,S. Saikumar, R.Mageshvaran, T.Narendiranath Babu, Application of Bacterial Foraging and Firefly Optimization Algorithm to Economic Load Dispatch Including Valve Point Loading".

# *CHAPTER-4*

## Hybridization of Differential Evolution and bacterial Foraging Optimization Algorithm:-

### 4.1 Introduction:-

Evolutionary optimization technique has been one of the major optimization techniques in past decade. Few key features of the evolutionary algorithms (EAs) include parallel computation, holistic learning approach and self-adaption. The application of evolutionary algorithms has been carried out extensively for solving constrained problems and practical engineering problems also. Though the EAs are widely used, in practice they deliver only marginal performance. Hence the current aim of researchers is to apply the complementary algorithms to enhance their performance. A current trend in the field of optimization using evolutionary algorithms tends to hybridize two or more algorithms in order to perform better than the individual algorithm. Though the population based search methods explore search spaces effectively with problems having high dimensionality, non convexity; they may be stuck in local optima and may not always provide the global optimum. Many real world large scale optimization problems may not provide acceptable solutions, when applied independently as seen in cases of combinatorial optimization. Thus instead of using traditional nature inspired algorithms there has been a increase in combination of algorithmic ideas since there is not a single strategy which can be used to solve all kinds of optimization problems[1]. Any superior performance in one class of problems generally results in inferior performance over another class. The combination of such algorithms is called hybridization or hybrid meta-heuristics. They combine the advantages of individual algorithms while overcoming individual weaknesses. Such an approach ensures that at least one of the algorithms provides optimum solution to each particular class of problems. The main concern while using hybridization is that the percentage of successful convergence to global optimum should increase as opposed to those obtained by standalone algorithm. several approaches of heuristic algorithms have been used to enhance the performance of EAs. Zmuda et al. [2] introduced a hybrid evolutionary learning scheme for synthesizing multi-class pattern recognition systems. Wang [3] developed a hybrid approach to improve the performance of EAs for a simulation optimization problem. A hybrid technique that combines GA and PSO, called genetic swarm optimization (GSO), was proposed by Grimaldi et al. [4] for solving an electromagnetic optimization problem. Li and Wang et al. [5] proposed a hybrid PSO using Cauchy mutation to reduce the probability of trapping local optima for PSO. Li and Yang [6] used a hybrid evolutionary algorithm base on Particle swarm optimization (PSO), Fast Evolutionary Programming (FEP), and Estimation of Distribution Algorithm (EDA) was used by them. Bashir and Neville [7] propose evolutionary computation algorithm featuring a novel adaptive elitism strategy and a sequential quadratic programming algorithm; combined in a collaborative portfolio with a validation procedure. Shi et al [8] prl and series forms.

## 4.2 Proposed method of Hybridization:-

The radical reduction in the computational time in the recent past coupled with the increasing demand to solve complex real world problems has enhanced the quest for more proficient nature-inspired metaheuristics. It is to be noted that two fundamental processes drive the evolution of an Evolutionary Algorithm (EA) population— the diversification process, which enables exploring different regions of the search space and the intensification process, which ensures the exploitation of previous knowledge about the fitness landscape. The effects of such exploration and exploitation processes need to be competently balanced by an EA for its qualitative performance both in computational complexity and run-time accuracy over different fitness landscapes. However the superiority of an EA in optimizing objective functions is subjected to the No Free Lunch Theorem (NFLT) [9]. According to NFLT the expected effectiveness of any two traditional EAs across all possible optimization problems is identical. A self-evident implication of NFLT is that the elevated performance of one EA, say A, over other EA, say B, for one class of optimization problems is counterbalanced by their respective performances over another class. It is therefore practically difficult to devise a universal EA that would solve all the problems. This apparently paves the way for hybridization of EAs with other optimization strategies, machine learning techniques and heuristics. In evolutionary computation paradigm, hybridization [10] refers to the process of integrating the attractive features of two or more EAs synergistically to develop a new hybrid EA. The hybrid EA is expected to outperform its ancestors both in accuracy and complexity over application-specific or general benchmark problems. The fusion of EAs through hybridization hence can be regarded as the key to overcome their individual limitations. In this paper, we propose a simple yet very powerful hybrid EA by collegially coalescing the attributes of two global optimizers— traditional Differential Evolution (DE) [11] and traditional Bacterial Foraging Algorithm (BFA) [12]. In our proposed hybridization stratagem, the chemotactic movement of bacteria is embodied into the modified version of mutation policy of DE to utilize the composite benefit of the explorative and exploitative capabilities of both ancestor algorithms. The chemotaxis of bacteria around their own positions provides them the local exploitation capability. On the other hand, DE/rand/1 mutation strategy offers DE a potential for global exploration. These facts have motivated us to propose a new hybrid algorithm, named Differential Evolution with Bacterial Chemotaxis (DEBC). In DEBC, the intensification process is controlled by the chemotactic movement of bacteria in its local neighbourhood, while the diversification is influenced by the DE mutation policy. As mentioned in earlier chapters, an evolutionary algorithm works through 4 steps, i.e. initialization, mutation,crossover/recombiantion,selection. In this proposed hybridization technique, the chemotaxis step of the Bacterial Foraging algorithm is introduced in crossover/recombination step of DE as shown in Pseudo code to achieve desired optima.

## 4.3 Pseudo Code:-

Let $x$ is a n-dimensional vector & f is real function of real valued arguments, three parameters of DEA are CR (defining crossover and mutation operations that are mutually exclusive), F (scaling factor of the difference of two individuals) and NP (population size) to generate the evolutionary process for $n$-dimensional problem, number of generation is $G_{max}$ D dimension, three randomly chosen individuals with index r1,r2 & r3, $C_i$ is step size taken in the random direction specified by the tumble, the bacterium will generate the tumble direction $\Delta_i$,

where $\vec{\Delta}_i = [\vec{\Delta}_{i1}, \vec{\Delta}_{i2}, \ldots\ldots \vec{\Delta}_D]$ .Thus,in computational chemotaxis the movement of the bacterium may be represented by-

$$\vec{X}prime \leftarrow \vec{X}prime + STEPsize \times \frac{\vec{\Delta}}{\vec{\Delta}X\vec{\Delta}^T}$$

```
1  Begin
2  G=0
3  Create a random initial population
4  for i = 1 to NP do
5   for j = 1 to D do
6          $x_{j,i}^{(G=0)} = x_j^{min} + rand_j[0,1].(x_j^{max} - x_j^{min})$
7      end for
8    end for
9  Evaluate Fitness Function for each individual of population
10    for i = 1 to NP do
11                 $f( x_i^{(G=0)})$
12   end for
13  Test vector generation
14   for G = 1 to MaxGen do
15     for i = 1 to NP do
16  Select Randomly  r1,r2,r3 ∈ [1,NP],r1≠r2≠r3≠i
17  Mutation & Crossover Process
18        jrand = randInt[1:D]
19        for j = 1 to D do
20              if(rand[0,1]<CR or j == jrand)then
21                    $v_{i,j}^{(G+1)} = x_{ir}^{(G)} + F*(x_{ir2}^{(G)} - x_{ir3}^{(G)})$
22          else
23                $v_{i,j}^{(G+1)} = x_{ij}^{(G)}$
24  Chemotaxis
25  Randomly initialize a vector $\vec{\Delta} = [\Delta_1, \Delta_2, .... \Delta_D]$ where $\Delta_j$ is uniformly distributed
within [-1,1] for j=[1,2,….,D];
26  $\vec{X}\_prime \leftarrow \vec{X}\_prime + STEP\_size \times \frac{\vec{\Delta}}{\vec{\Delta} X \vec{\Delta}^T}$
```

```
27 Evaluate $f(\vec{X}\_prime)$;
28 m←0;
29 while m< $N_s$ do
30 Begin
31 m←m+1;
32   if f($\vec{X}\_prime$) < last_cost
33     Then do
34     Begin
35         last_cost←f($\vec{X}\_prime$);
36         $\vec{X}\_prime$←$\vec{X}\_prime$ + STEP_size x $\frac{\vec{\Delta}}{\vec{\Delta}X\vec{\Delta}^T}$ ;
37           Evaluate $f(\vec{X}\_prime)$;
38       Endif
39     Endfor
40   Endfor
41 Selection
42       if(f($v^{(G+1)}$≤ f($x_i^{(G)}$)) then
43           $x_i^{(G+1)}$= $v_{i,j}^{(G+1)}$
44       else
45           $x_i^{(G+1)}$= $x_i^{(G)}$
46     end if
47    end for
48   end for
49 End
```

## 4.4 Experiments and results:-

***Benchmark functions:***

The proposed algorithm is tested with the 6 benchmark functions of popular 25 benchmark functions given in CEC-2005. The functions are described below-

*1. Rastrigin Function-*

This function, the so-called Rastrigin's function , is an example of a highly multimodal search space. It has several hundred local optima in the interval of consideration. The function can be formulated as,

$$f(x) = A_n + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)]$$

*where A = 0*

*Global minima f(0,……,0)=0, Search Domain: -5.12≤* $x_i$ *≤ 5.12*

*2. Sphere Function-*

The function sphere, *f1*, can be said as the center point of every optimization algorithm. It is a smooth, unimodal, and symmetric function and it does not present any of the difficulties that we have discussed so far. The performance on the sphere function is a measure of the general efficiency of an algorithm. Below, the function is shown,

$$f(x) = \sum_{i=1}^{n} x_i^2$$

*Global minima* $f(x_1,\ldots,x_n) = f(0,\ldots,0) = 0$

*Search domain:* $-\infty \leq x_i \leq +\infty$

*3. Ackley Function-*

This is a highly multimodal function.The function is not convex. It is non-separable & Differentiable. The function can be represented by-

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

*Global minima f(0,0)=0 Search Domain -5≤ x,y ≤5*

*4. Beale Function-*

This is a continuous , non convex, multimodal function. The function can be defined on any input domain but it is usually evaluated on x∈ [-4.5,4.5] ∀ i=1,2.

The function can be defined as,

$$f(x,y)=(1.5-x-xy)^2 + (2.25-x-xy^2)^2 + (2.625-x+xy^3)^2$$

*Global minima* $f(3,0.5)=0$ *Search Domain* $-4.5 \leq x,y \leq 4.5$

*5. Matyas Function-*

Another unimodal function is Matyas function. The other properties of this test function are, it is convex, differentiable, continuous, non separable. The function can be represented as-

$$f(x,y) = 0.26(x^2+y^2)-0.48xy$$

*Global minima* $f(0,0)=0$ *Search domain* $-10\leq x,y\leq 10$

*6. Booth Function-*

This function is continuous, convex, unimodal & differentiable. The function is formulated as-

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2$$

*Global minima* $f(1,3)=0$ *Search domain* $-10\leq x,y\leq 10$

***Results:***

The six numbers benchmark functions are used to test the proposed algorithm. Here, the code has been executed for 200 iterations. The table below represents the value of fitness for 200th iteration.

| Function Name | DE | DEBC |
|---|---|---|
| Rastrigin | 0.000000 | 0.3752540 |
| Beale | 5.9165e-31 | 0.0006447 |
| Sphere | 1.2794e-58 | 0.0033740 |
| Booth | 7.8886e-31 | 0.0106448 |
| Matyas | 2.1303e-53 | 0.0000466 |
| Ackley | -5.119100 | -5.1154206 |

*Table 4.1: DE & DEBC fitness comparison*

From the above table we can conclude that the fitness value for each test function become greater while tested by the proposed hybrid algorithm than ordinary differential evolution. The application of proposed hybrid algorithm is shown for Rastrgin Function upto 20th iteration-

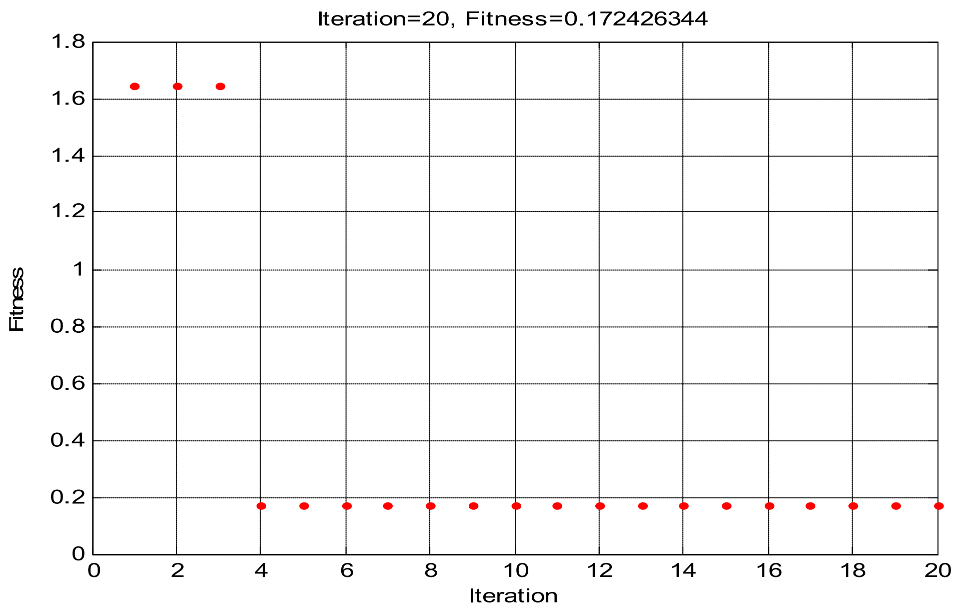


*Fig.4.1: Ordinary Differential evolution Iteration vs Fitness plot for Rastrigin Function*

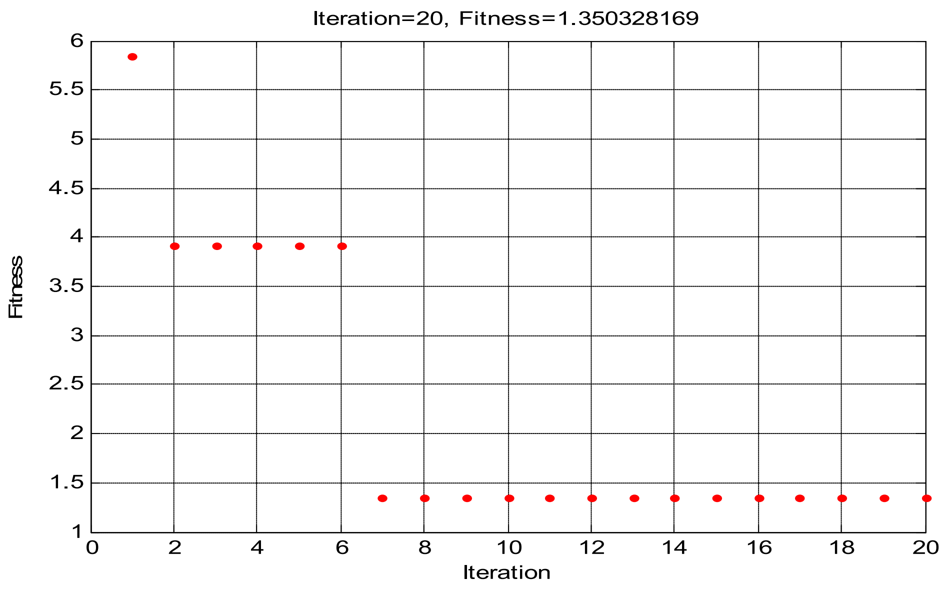


*Fig.4.2: Hybrid Differential evolution Iteration vs Fitness plot for Rastrigin Function*

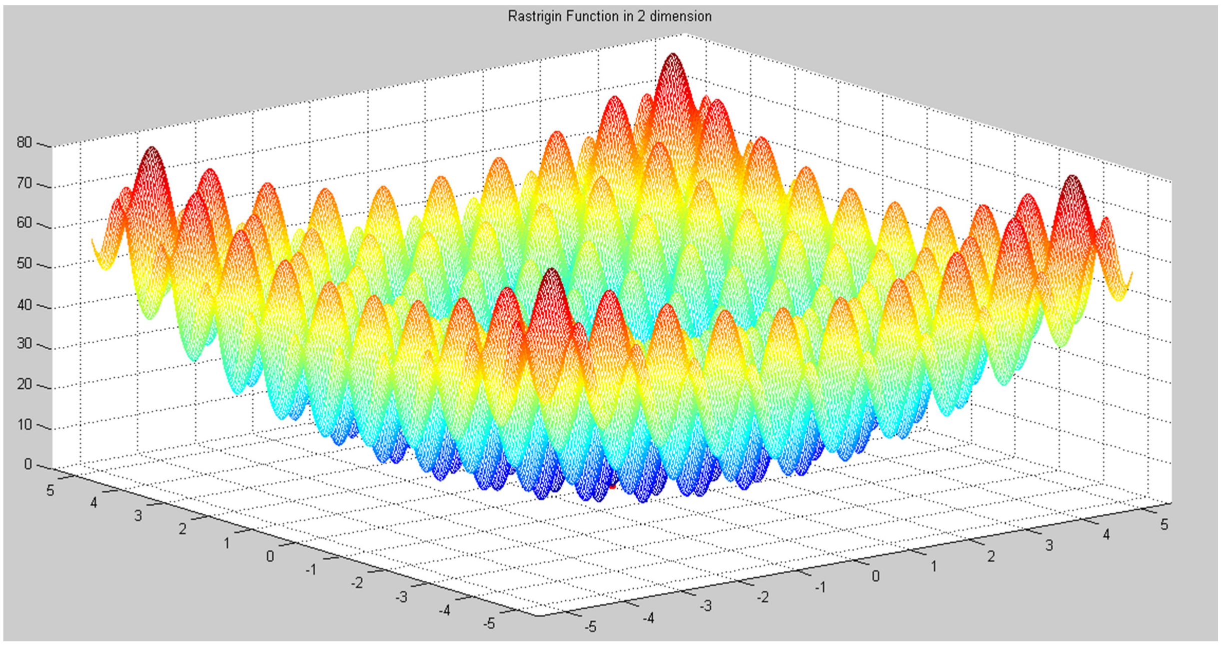


*Fig.4.3: 3 Dimensional plot for Rastrigin Function*

## *Refernces:-*

[1] D.H. Wolpart,"No free Lunch Theorems for search".

[2] M.A.Zmuda, M.M.Rizki, L.A.Tamburino, "Hybrid Evolutionary learning for synthesizing multi class pattern recognition systems". Applied Soft Computing, Vol.2, Issue 4, p.269-282, February 2003.

[3] L.Wang, D.Zheng, "An effective hybrid optimization strategy for job scheduling problems". Computer and Operations Research, Vol.28, Issue.6, p.585-596, May 2001.

[4] E.A.Grimaldi, F.Grimaccia, M.Mussetta, P.Pirinoli, "Genetical Swarm Optimization: a New Hybrid Evolutionary Algorithm for Electromagnetic Applications". 18th International Conference on Applied Electromagnetics and Communications, ICECom, Oct. 2005.

[5] E.A.Grimaldi, F.Grimaccia, M.Mussetta, P.Pirinoli, "Genetical Swarm Optimization: a New Hybrid Evolutionary Algorithm for Electromagnetic Applications". 18th International Conference on Applied Electromagnetics and Communications, ICECom, Oct. 2005.

[6] C.Li, S.Yang, "An Island Based Hybrid Evolutionary Algorithms for Optimization". Simulated Evolution and Learning Lecture Notes in Computer Science, Vol.5361, p.180-189, 2008.

[7] H.A.Bashir, R.S.Neville, "A Hybrid Evolutionary Computation Algorithms for Global Optimization". WCCI 2012 IEEE World Congress on Computational Intelligence, 2012.

[8] X.H.Shi, Y.H.Lu, C.G.Zhou, H.P.Lee, W.Z.Lin and Y.C.Liang, "Hybrid Evolutionary Algorithms Based on PSO and GA". The 2003 Congress on Evolutionary Computation, 2003, p.2393-2399 Vol.4.

[9] David H. Wolpart, William G. Macready, 'No free lunch theorem for search',SFI-TR-95-02-010, The Santa Fe Institute, February 23,1996

[10] A. Sinha and D. Goldberg. A survey of hybrid genetic and evolutionary algorithms. Technical Report 2003004, Illinois Genetic Algorithms Laboratory (IlliGAL), January 2003.

[11] Kenneth V. Price ,Rainer M. Storn Jouni A. Lampinen, ‘Differential Evolution- A Practical Approach to Global Optimization’ , Springer-Verlag Berlin Heidelberg 2005.

[12] Kevin M. Passino,”Biomimicry for Optimization, Control, and Automation”, Springer-Verlag London Limited 2005 .

# *CHAPTER-5*

## Conclusion & Future Research Direction

*This chapter briefly highlights the findings & contributions of the thesis. It briefly discusses the real life applications of the hybridization of two optimization algorithms i.e., differential evolution & bacterial foraging. It also introduces some of the future works for interested readers that may be carried out in extension to the algorithms that have already been developed.*

## 5.1 Conclusion :

Generally, heuristic algorithms are used to enhance the performance of evolutionary algorithm to achieve the global optima.
In various cases it is observed that the steps of a particular evolutionary algorithm which is giving an optimal value for a particular problem is giving undesired value in another problem, thus the concept of hybridization was introduced.
There are two prominent issues of EAs in solving global and highly nonconvex optimization problem. These are: (i) Premature convergence: The problem of premature convergence results in the lack of accuracy of the final solution. The final solution is a feasible solution close to the global optimal, often regarded as satisfactory or close-to-optimal solution. (ii) Slow convergence: Slow convergence means the solution quality does not improve sufficiently quickly. It shows stagnation or almost flat on a convergence graph (either a single iteration or the average of multiple iterations).
Hybrid algorithms are two or more algorithms that run together and complement each other to produce a profitable synergy from their integration
Chapter 4 has shown how these two optimization algorithms are hybridized.

The boundary condition checking after the mutation step in differential evolution algorithm is hybridized by bacterial foraging algorithm. Thus we found our desired performance level.

## 5.2 Future research direction:-

- The proposed hybrid algorithm is tested on six CEC 2005 benchmark functions. The same algorithm can be tested over other benchmark functions, like – Holder Table function, Egg holder function, Cross-in-tray function, Himmelblau's function, Schaffer function etc.
- Every search algorithm needs to address the exploration and exploitation of a search space. Exploration is the process of visiting entirely new regions of a search space, whilst exploitation is the process of visiting those regions of a search space within the neighborhood of previously visited points. In order to be successful, a search algorithm needs to establish a good ratio between exploration and exploitation. Further study of

exploration & exploitation through population diversity can be done in the field of evolutionary algorithm.

- Moreover, the study can be done on hybridization of more than two number of optimization algorithms.

Many new algorithms have been developed in recent years. For example, the bio-inspired algorithms such as Artificial Bee Colony Algorithm (ABC)[1], Bat Algorithm (BA)[2], Cuckoo Search (CS)[3], Firefly Algorithm (FA)[4], Flower Pollination Algorithm (FPA)[5], Glowworm Swarm Algorithm (GlowSA)[6], Hunting Search Algorithm (HSA)[7], Eagle Strategy (ES)[8], Roach Infestation Optimization (RIO)[9], Gravitational Search Algorithm (GravSA)[10], Artificial Fish School Algorithm (AFS)[11], Artificial Plant Optimization Algorithm (APO)[12], Krill Herd Algorithm (KHA)[13] and others.

These algorithms may possess entities and some novel characteristics for hybridization that remain to be discovered in the near future.

In many works, hybrid algorithms seem to improve results in terms of the overall convergence speed and accuracy. However, these convergence graphs are often plotted with respect to the number of iterations. This simply means that the faster convergence does not mean the true convergence rate because the hybrid usually uses a higher number of (internal or implicit) iterations. For example, for collaborative (sequential type) hybrid algorithm such as GA-PSO, a cycle, or one iteration comprises GA and PSO. For a fair comparison, this should be considered as two cycles instead of one in the convergence graph. To avoid this issue, the final run time should be utilized as a metric when comparing a hybrid algorithm with non-hybrid algorithms.

***Refernces:-***

[1] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, Nurhan Karaboga, “A comprehensive survey: artificial bee colony (ABC) algorithm and applications Published online: 11 March 2012 © Springer Science+Business Media B.V. 2012

[2] Xin-She Yang,A New Metaheuristic Bat-Inspired Algorithm” . NICSO 2010, SCI 284, pp. 65–74, 2010. springerlink.com © Springer-Verlag Berlin Heidelberg 2010

[3] Xin-She Yang, Suash Deb, Cuckoo Search via L´evy Flights” , 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC 2009)IEEE

[4] Xin-She Yang,  Firefly algorithm, stochastic test functions and
design optimization”. Int. J. Bio-Inspired Computation, Vol. 2, No. 2, 2010”

[5] Mohamed Abdel-Basset,Laila A. Shawky, Flower pollination algorithm: a comprehensive review”, Springer Science+Business Media B.V., part of Springer Nature 2018.

[6] HE Deng-xu, Liu Gui-qing, ZHU Hua-zheng,Glowworm swarm optimization algorithm for solving multi-objective optimization problem”. 2013 Ninth International Conference on Computational Intelligence and Security IEEE

[7] R. Oftadeh, M. J. Mahjoob,A new meta-heuristic optimization algorithm: Hunting Search”, Center for Mechatronics and Automation, School of Mechanical Engineering, University of Tehran, Tehran, Iran, 9781-4244-3428-2/09/2009 IEEE

[8] Xin-She Yang, Suash Deb, Xingshi He, Eagle Strategy with Flower Algorithm”. 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI),IEEE.

[9] Timothy C. Havens, Christopher J. Spain, Nathan G. Salmon, James M. Keller, Roach Infestation Optimization”, 2008 IEEE Swarm Intelligence Symposium St. Louis MO USA, September 21-23, 2008.

[10] Norlina Mohd Sabri, Mazidah Puteh, Mohamad Rusop Mahmood, An Overview of Gravitational Search Algorithm Utilization in Optimization Problems”, 2013 IEEE 3rd International Conference on System Engineering and Technology, 19 - 20 Aug. 2013, Shah Alam, Malaysia

[11] Jie Hu, Xiangjin Zeng, Jiaqing Xiao, Artificial Fish School Algorithm For Function Optimization”,IEEE, 30 December 2010, Wuhan, China

[12] Ziqiang Zhao, Zhihua Cui, Jianchao Zeng, Xiaoguang Yue, Artificial Plant Optimization Algorithm for Constrained Optimization Problems", 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications,IEEE, Shenzhan, China.

[13] Amir Hossein Gandomi, Amir Hossein Alavi, Krill herd: A new bio-inspired optimization algorithm", Elsevier,2012