CHAPTER 1

INTRODUCTION

Engineering problems can be defined as search for one near optimal description among many possibilities, under real time constraints. Search techniques such as Bacterial foraging, Particle swarm and Firefly based optimization algorithms are now among the latest research topics. This work throws light on these bio-inspired evolutionary algorithms that are used to optimize the objective function, under given constraints. Optimization as an engineering problem has been exposed as their one application along with the several advancements made in this field. These techniques are explained in detail along with work and research that has been done in that area. It also provides the insight of modification and improvements, these algorithms continue to enjoy thus performing better for a specified problem.

Optimization has been an active area of research for several decades. As many real-world optimization problems become increasingly complex, better optimization algorithms are needed. Over the past few decades, many biologically inspired computational methodologies have been invented, such as evolutionary algorithms which include genetic algorithms, bacterial foraging, particle swarm firefly algorithms and many others. They tend to enjoy a regular attention by scientist in several research areas of soft computing and others as well.

Particle swarm optimization algorithm, first proposed by Dr. Kennedy and Dr. Eberhart in 1995 is a new intelligent optimization algorithm developed in recent years [1], which simulates the migration and aggregation of bird flock when they seek for food. This algorithm adopts a strategy based on particle swarm and parallel global random search. PSO algorithm determines search path according to the velocity and current position of particle without more complicated evolution operation. Particle swarm optimization is a high-performance optimizer

that is very easy to understand and implement. It is similar in some ways to genetic algorithms or evolutionary algorithms, but requires less computational bookkeeping and generally only a few lines of code [1]. Particle swarm optimization originated in studies of synchronous bird flocking and fish schooling, when the investigators realized that their simulation algorithms possessed an optimizing characteristic[2]-[4].

A particularly interesting group behaviour[8], which can be translated into efficient optimization algorithms, has been demonstrated for several motile species of bacterial colonies such as Samonellatyphimurium, where intricate stable spatio-temporal patterns based on stimuli of cell-cell signalling and foraging are formed in semi-solid nutrient media.

Natural selection tends to eliminate animals with poor "foraging strategies" (methods for locating, handling, and ingesting food) and favour the propagation of genes of those animals that have successful foraging strategies since they are more likely to enjoy reproductive success (they obtain enough food to enable them to reproduce) [8]. After many generations, poor foraging strategies are either eliminated or shaped into good ones (redesigned). Vast applications have been found where BFO has shown remarkable results and has been modified for different problems according to the objective function. Initial applications of evolutionary algorithm were meant for static optimization problems but in recent years the emergence of another member of the EA family[15] bacterial foraging algorithm (BFA), the self-adaptability of individuals in the group searching activities has attracted a great deal of interests including dynamic problems. W. J. Tang and Q. H. Wu have contributed their work by proposing DBFA, which is especially designed to deal with dynamic optimization problems, combining the advantage of both local search in BFA and a new selection scheme for diversity generating. They used the moving peaks benchmark (MPB) [13] as the test bed for experiments. The performance of the DBFA is evaluated in two ways. The first is concerned with the

convergence of the algorithm in random periodical changes in an environment, which are divided into three ranges from a low probability of changes to a higher one.

Mechanisms of firefly communication via luminescent flashes and their synchronization has been imitated effectively in various techniques of wireless networks design, dynamic market pricing and mobile robotics [19], [25]-[27]. The flashing light of fireflies is an amazing sight in the summer sky in tropical and temperate regions there are about two thousand firefly species, and most firefly produce short and rhythmic flashes. Many bio-inspired algorithm exist that use simple rules and local information to create a global consensus of a single parameter, such as clock time in the flashing firefly algorithm, or location in the slime algorithm. In this paper we identify a control loop as the core structure of several bio-inspired algorithms. Thresholding is one of the most important techniques for performing image segmentation. It is generally simple and computationally efficient. The MEFFT algorithm simulates the behaviour of fireflies and the phenomenon of bioluminescent communication to develop the algorithm to select the adequate thresholds for image segmentation. The segmentation results of MEFFT algorithm are promising and it encourages further researches for applying this algorithm to complex and real-time image analysis problems such as target recognition, complex document analysis and biomedical image application [19]. The other application include synchronisation of wireless network which be seen in [25]. An application in pulse-coupled oscillator's model addresses the issue of synchronization of the oscillators with different frequencies [27]. As a continuation of this research, it is important to study how the topology and communication delay of the distributed oscillators system affect the synchronization process. Fireflies, also called lighting bugs, are one of the most special and fascinating creatures in nature. These nocturnal luminous insects of the beetle family *Lampyridae* (order *Coleoptera*), inhabit mainly tropical and temperate regions, and their population is estimated at around 1900 species. Bioluminescent signals are known to serve as

elements of courtship rituals, methods of prey attraction, social orientation or as a warning signal to predators (in case of immature firefly forms commonly referred to as glow worms). The phenomenon of firefly glowing is an area of continuous research considering both its biochemical and social aspects.

Problem statement: *T*his work contains the study of meta-heuristic algorithms and their performance on benchmark problems. Here Comparative Analysis on the performances of the algorithms on the basis of parameters like elapsed time, mean and standard deviations is done; this brings the best algorithm to light. BFA is used in indirect adaptive control of two control applications, namely Liquid level nonlinear control system and DC servomotor. Here algorithm aims to search the best member (in terms of controller) which tracks the desired trajectory.

CHAPTER 2

REVIEW OF META-HEURISTIC ALGORITHMS

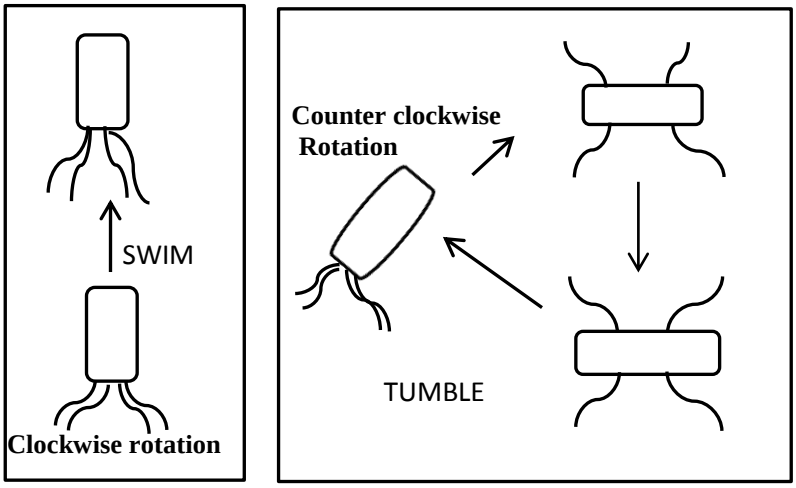
In artificial intelligence, an evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based meta-heuristic optimization algorithm. An EA uses some mechanisms inspired by biological evolution: reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the environment within which the solutions "live" (see also cost function). Evolution of the population then takes place after the repeated application of the above operators. Artificial evolution (AE) describes a process involving individual evolutionary algorithms; EAs are individual components that participate in an AE.

Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by successes in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry..

Bio-inspired algorithms and systems are routinely applied to hard and large problems in a variety of areas. Some examples are optimization problems solved with genetic algorithms, routing strategies inspired by honey bee behaviour, resource discovery and data mining computations in Grid, Cloud and P2P frameworks, achieved by ant-inspired algorithms, and so on. Some other bio-inspired algorithms are bee algorithm, simulated annealing, harmony search, bacterial foraging algorithms, ant colony optimization, particle swarm optimization and firefly algorithm. Here in this work three algorithms have been taken for analysis.

2.1 BACTERIAL FORAGING ALGORITHM

Bacterial foraging algorithm is inspired by the pattern exhibited by bacterial foraging behaviours [1]. They tend to maximise energy per unit time. Bacteria have the tendency to gather to the nutrient-rich areas by an activity called "chemotaxis". It is known that bacteria swim by rotating whip-like flagella driven by a reversible motor embedded in the cell wall. *E. coli* has 8-10 flagella placed randomly on a cell body. When all flagella rotate counter clockwise, the move is called *Run*. When the flagella rotate clockwise, they all pull on the bacterium in different directions, which causes the bacteria to *Tumble* as shown in figure 1.1



A. Swim B. Tumble

Fig. 2.1 Swim(A) and tumble(B) of a bacterium

The foraging algorithm is based on the behaviour of the E. coli bacteria, which has tree fundamental steps: chemo taxis, reproduction, and elimination and dispersion. Chemo taxis is used in order to move and search for food, this step has much resemblance with a biased random-walk model [2].

In order to maximise energy in minimum time bacteria tend to follow different search strategies as described in figure 1.2

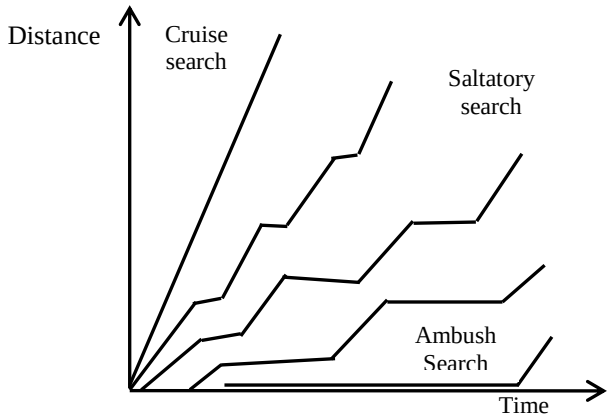


Fig. 2.2 Search strategies for foraging bacteria

The chemotactic operator employed in BFOA is supposed to guide the swarm to converge toward optima. Reproduction is used to make an identical copy; and elimination and dispersion is used to avoid noxious elements that can kill or spread a group of bacteria in the search space.

When bacteria get food in sufficient amount, they increase in length, and in presence of suitable temperature, they break in the middle to form an exact replica of themselves. This phenomenon inspired Passino [1] to introduce an event of reproduction in BFOA. Due to the occurrence of sudden environmental changes or attack, the chemotactic progress may halt, and a group of bacteria may move to some other places. This constitutes the event of elimination–dispersal in the real bacterial population, where all the bacteria in a region are killed or a group is dispersed into a new part of the environment. As said, bacteria try to move to areas which are rich in nutrients and free of noxious substances [4]. If θ is a position and $J(\theta)$ is a function, the more the negative the more the nutrients prevail over the noxious substances, then we can imagine that bacteria naturally try to find those, where $J(\theta)$ has its minima.

A bacterium position after a tumble can be determined through equation (2.1), where the position at a given instant is calculated in terms of the position at the previous instant and the step size *C(i)* applied in a random direction $\phi(j)$, generated by the bacterium tumble

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i) * \phi(j,k,l) \quad (2.1)$$

Where *i* indicate the bacterium in a total population of *S* individuals, *j* the chemotactic step, the reproductive step *k*, and the elimination and dispersal step *l*.

The health function is calculated at these positions of bacteria as (2.2)

$$J(i,j,k,l) = J(i,j,k,l) + Jcc(\theta i(j,k,l), P(j,k,l)). \quad (2.2)$$

Here second expression shows the effect of attraction and repulsion among the bacteria. The exact expression of

$J_{cc}(\theta^i(j,k,l), P(j,k,l))$ is in [1].

And this then is compared with the previous health function as

$$J\ (\theta^i(j+1,k,l)) < J(\theta^i(j,k,l)) \quad (2.3)$$

If it's true then the bacterium will keep moving in the same direction ("run" behaviour) for a given maximum number of steps, after which a "tumble" will occur anyway. 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. This is done through reproduction step. Some bacteria are liquidated at random with a very small probability while the new replacements are randomly initialized over the search space.

i. Pseudo code:

1. *INITIALIZE PARAMETERS*: n, N, Nc, Ns, Nre, Ned, Ped,

C(i)(i=1,2…N), i =1,2,---

Where,

n: Dimension of the search space,

N: The number of bacteria in the population,

NC : Chemotactic steps,

Nre: The number of reproduction steps,

Ned : the number of elimination-dispersal events,

Ped: Elimination-dispersal with probability,

C (i): the size of the step taken in the random direction specified by the tumble.

2. *ELIMINATION-DISPERSAL* LOOP: *l*=*l*+1

3. *REPRODUCTION LOOP*: *k*=*k*+1

4. *CHEMO TAXIS LOOP*: *j*=*j*+1

[a] For *i* =1,2…N, take a chemotactic step for bacterium *i* as follows.

[b] Compute fitness function, J (*i, j, k, l*).

Let,$J\ (i,j,k,l)\ = J\ (i,j,k,l) + J_{cc}(\theta^{i}(\ j,k,l),P(\ j,k,l))$

(i.e. add on the cell-to cell attractant–repellent profile to simulate the swarming behaviour) .Where, J_{cc} is defined in .

[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 $\Delta\ (i) \in Rn$ with each element $\Delta m(i), m = 1,2,\ldots,p,a$ random number on [-1, 1].

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

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 $J(i,j,k,l) = J(i,j,k,l) + Jcc(\ \theta i(\ j,k,l),P(\ j,k,l))$

[g] Swim

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

ii) While $m < N_S$ (if have not climbed down too long).

• Let $m = m + 1$.

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

J_{last}= $J\ (i, j + 1, k, l)$ and

Let $\theta^{\iota}(J + \dot{1}, k, l) = \theta(j, k, l) + c(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$

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

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

[h] Go to next bacterium (*i*, 1) if i ≠ *N* (i.e., go to [b] to process the next bacterium).

5. If *j*<*Nc*, go to *Step 3*. In this case, continue chemo taxis, since the life of the bacteria is not over.

6. *REPRODUCTION:*

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

Let $J^i_{health} = \sum_{j=1}^{N_c} J(i, j, k, l)$

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* (higher cost means lower health).

[b] The *Sr* bacteria with the highest J_{health} values die and the remaining *Sr* 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).

7. If *k* < *Nre*, 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.

8. *ELIMINATION-DISPERSAL*: For i=1, 2..., N, with probability Ped, eliminate and disperse each bacterium, and this result in keeping the number of bacteria in the population constant. To do this, if a bacterium is eliminated, simply disperse one to a random location on the optimization domain. If $l<Ned$, then go to *Step. 2*; otherwise end

ii. Advancements and Applications of BFO

Vast applications have been found where BFO has shown remarkable results and has been modified for different problems according to the objective function. Initial applications of evolutionary algorithm were meant for static optimization problems but in recent years the emergence of another member of the EA family[5]– bacterial foraging algorithm (BFA), the self-adaptability of individuals in the group searching activities has attracted a great deal of interests including dynamic problems. W. J. Tang and Q. H. Wu have contributed their work by proposing DBFA, which is especially designed to deal with dynamic optimization problems, combining the advantage of both local search in BFA and a new selection scheme for diversity generating. They used the moving peaks benchmark (MPB) [6] as the test bed for experiments. The performance of the DBFA is evaluated in two ways. The first is concerned with the convergence of the algorithm in random periodical changes in an environment, which are divided into three ranges from a low probability of changes to a higher one. The second is testing a set of combinations of the algorithm parameters which are largely related to the accuracy and stability of the algorithm. All results are compared with the existing BFA [1], and show the effectiveness of DBFA for solving dynamic optimization problems.

It is worth mentioning that the diversity of DBFA changes after each chemotactic process rather than the dispersion adopted by the BFA after several generations. The DBFA utilizes not only the local search but also applies a flexible selection scheme to maintain a suitable diversity during the whole evolutionary process. It outperforms BFA in almost all dynamic

environments. The results are shown in [5]. They have further given solution for global optimization given in [7].

The novel BSA has been proposed for global optimization. In this algorithm, the adaptive tumble and run operators have been developed and incorporated, which are based on the understanding of the details of bacterial chemotactic process. The operators involve two parts: the first is concerned with the selections of tumble and run actions, based on their probabilities which are updated during the searching process; the second is related to the length of run steps, which is made adaptive and independent of the knowledge of optimization problems. These two parts are utilized to balance the global and local searching capabilities of BSA. Beyond the tumble and run operators, attraction and mutation operations have also been developed.

A.ABRAHAM, A. BISWAS, S. DASGUPTA AND S. DAS have shown [8] that the major driving forces of Bacterial Foraging Optimization Algorithm (BFOA) is the reproduction phenomenon of virtual bacteria each of which models one trial solution of the optimization problem.

BFO and PSO have been used in combination and their combined performance has been utilised to incorporate the merits [19] of two bio-inspired algorithms to improve the convergence for high-dimensional function optimization. It is assumed that the bacteria have the similar ability like birds to follow the *best bacterium* (bacterium with the best position in the previous chemotactic process) in the optimization domain. The position of each bacterium after every move (*tumble* or *run*) is updated according to (3)

$$\theta^i(j+1,k,l) = \theta^i(j+1,k,l) + Ccc(\theta^b(j,k,l) - \theta^i(j,k,l), \qquad (2.4)$$

if $J^i(j+1,k,l) > J_{min}(j,k,l)$

where $\theta^b(j,k,l)$ and $J_{min}(j,k,l)$ are the position and fitness value of the *best bacterium* in the previous chemotactic process respectively, C_{cc} is a new parameter, called *attraction factor*, to adjust the bacterial trajectory according to the location of the *best bacterium*.

2.2 PARTICLE SWARM OPTIMIZATION ALGORITHM

The particle swarm concept originated as a simulation of a simplified social system. The original intent was to graphically simulate the graceful but unpredictable choreography of a bird flock. Initial simulations were modified to incorporate nearest-neighbour velocity matching, eliminate ancillary variables, and incorporate multidimensional search and acceleration by distance (Kennedy and Eberhart 1995, Eberhart and Kennedy 1995). At some point in the evolution of the algorithm, it was realized that the conceptual model was, in fact, an optimizer. Through a process of trial and error, a number of parameters extraneous to optimization were eliminated from the algorithm, resulting in the very simple original implementation (Eberhart, Simpson and Dobbins 1996).

Particle swarm optimization is recently invented high-performance optimizer that is very easy to understand and implement. The particle swarm optimizer shares the ability of genetic algorithm to handle arbitrary non-linear cost functions, but with much simpler implementation. Particle swarm optimization is inspired by the social behaviour of flocks of birds and schools of fish. A number of simple entities, particles are placed in the domain of definitions of some function or problem. The fitness- the value of optimization function – of each particle is evaluated at its current location. The movement of each particle is determined by its own fitness and of particles in its neighbourhood in the swarm.

A particle represents a potential solution. The velocity V_i^d and position X_i^d of the d^{th} dimension of the i^{th} particle are updated as follows (2.5) & (2.6).

$$V_i^d \leftarrow V_i^d + C1 * rand_i^d * (pbest_i^d - X_i^d) + C2 * rand2_i^d * (gbest_i^d - X_i^d) \quad (2.5)$$

$$X_i^d \leftarrow X_i^d + V_i^d \quad (2.6)$$

Where $X_i = (X_i^1, X_i^2, \dots, X_i^D)$ is the position of i^{th} particle $V_i = (V_i^1, V_i^2, \dots, V_i^D)$ represents velocity of the particle i. pbest =($pbest_i^1$, $pbest_i^2$, … , $pbest_i^D$) is the best previous position yielding the best fitness value for their i^{th} particle; and gbest = ($gbest_i^1$, $gbest_i^2$, … , $gbest_i^D$) is the best position discovered by whole population[14]. C_1 and C_2 and are the acceleration constants reflecting the weighting of stochastic acceleration terms that pull each particle toward pbest and gbest positions, respectively. $rand1_i^d$ and $rand2_i^d$ are two random numbers in the range [0, 1].

i. Pseudo code:

1. Generate the initial swarm by randomly generating the position and velocity for each particle;
2. Evaluate the fitness of each particle;
3. Repeat
4. for each particle *i* do
5. Update particle *i* according to (1) and (2);
6. If $f(x_i) < f(x_{pbesti})$ then
7. $x_{pbesti} = x_i$
8. if $f(x_i) < f(x_{gbest})$ then
9. $x_{gbest} := x_i$

10. End if

11. End if

12. End for

13. Until the stop criterion is satisfied

ii. Advancements and Applications of PSO

APPSO (Agent based parallel PSO) is based on two types of agents: one *coordination agent* and several *swarm agents*. The swarm is composed of various sub-swarms, one for each swarm agent. The coordination agent has administrative and managing duties. All the calculations are done by the swarm agents (see Figure 2.3).

In order to gain benefit from the large knowledge and insights achieved in the research field of sequential PSO it is important to modify the swarm's behaviour as little as possible. The inevitable changes to the algorithm due to the parallelization should also lead to positive effects with respect to solution quality. This is realized by using strategically niching where according to the requirements the sub-swarms either work together searching for one global optimum or spread over the search space to find various global/local optima, e.g. on multimodal functions.

To address the problem of space locus searching, a slowdown particle swarm optimization (SPSO) is proposed to improve the convergence performance of particle swarm from the position viewpoint.

The particle swarm in SPSO is divided into many independent sub-swarms to guarantee that particles converge to different position, since space locus has multiple optimal solutions and requires the convergence of both fitness and position of particle. Furthermore, particle velocity is updated by half according to fitness to achieve the position convergence [14].

Several PSO algorithms have been recently proposed to address DOPs of which using multi swarms seems a good technique. The multi swarm method can be used to enhance the diversity of the swarm, with the aim of maintaining multiple swarms on different peaks.

One of the reasons that particle swarm optimization is attractive is that there are very few parameters to adjust Particle swarm optimization has been used for approaches that can be used across a wide range of applications, as well as for specific applications focused on a specific requirement.

In this brief section, we cannot describe all of particle swarm's applications, or describe any single application in detail. Rather, we summarize a small sample.

Generally speaking, particle swarm optimization, like the other evolutionary computation algorithms, can be applied to solve most optimization problems and problems that can be converted to optimization problems. Among the application areas with the most potential are system design, multi-objective optimization, classification, pattern recognition, biological system modelling, scheduling (planning), signal processing, games, robotic applications, decision making, simulation and identification.

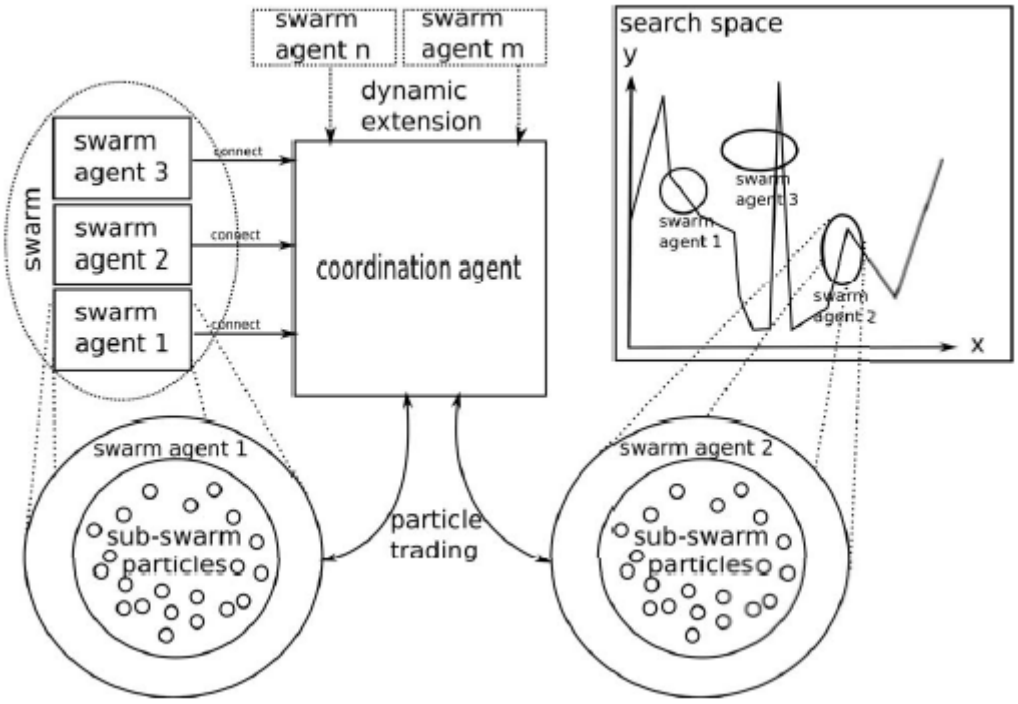


Fig. 2.3 Structure of APPSO network system [14]

Examples include fuzzy controller design, job shop scheduling, real time robot path planning, image segmentation, EEG signal simulation, speaker verification, time-frequency analysis, modelling of the spread of antibiotic resistance, burn diagnosing, gesture recognition and automatic target detection, to name a few.

2.3 *FIREFLY ALGORITHM*

The flashing light of fireflies is an amazing sight in summer sky in the tropical and temperate regions. There are about two thousand firefly species, and most fireflies produce short and rhythmic flashes. The pattern of flashes is often unique for a particular species. The flashing light is produced by a process called bioluminescence. However, two fundamental functions of such flashes are to attract mating partners and to attract potential prey. In addition it may also serve as a protective warning mechanism to remind potential predators of the bitter taste of fireflies.

We know that the light intensity at a particular distance *r* from the light source obeys the inverse square law. That is to say, the light intensity *I* decreases as the distance *r* increases in terms if $I \propto 1/r^2$. Furthermore, the air absorbs light which becomes weaker as the distance increases. These two combined factors make most fireflies visual to a limit distance, usually several hundred meters at night, which is good enough for fireflies to communicate.

The flashing light can be formulated in such a way that it is associated with the objective function to be optimized, which makes it possible to formulate new optimization algorithms.

Assumptions:

1. All fireflies are unisex so that one firefly will be attracted to the other fireflies regardless of their sex.

2. The brighter one will move towards the less bright one. The attractiveness is proportional to the brightness and they both decrease as the distance increase. And if there is no brighter one than a particular firefly, it will move randomly;

3. The brightness of a firefly is affected or determined by the landscape of the objective function.

Light intensity and the attractiveness:

In the firefly algorithm, there are two important issues: the variation of light intensity and formulation of the attractiveness. For simplicity, we can always assume that the attractiveness of a firefly is determined by its brightness which in turn is associated with the encoded objective function. In the simplest case particular location *x* can be chosen as *I(x) is directly proportional to f(x).* However attractiveness β is relative.

In simplest form, the intensity *I(r)* varies according to inverse square law

$$I(r) = \frac{I_s}{r^2}$$

Where I_s is the intensity at the source. For a given medium with light absorption coefficient γ, the light intensity I varies with the distance r. That is

$$I = I_0 e^{-\gamma r^2}, \quad (2.7)$$

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

$$I(r) = I_0 e^{-\gamma r^2} \quad (2.8)$$

Thus β of the firefly can be defined as

$$\beta = \beta_0 e^{-\gamma r^2} \quad (2.9)$$

Where β_0 is attractiveness at r=0

The movement of a firefly i is attracted to another more attractive firefly j is determined by

$$x_i = x_i + \beta_0 e^{-\gamma r_{ij}^2}(x_i - x_j) + \alpha\epsilon_i, \quad (2.10)$$

where second term is due to the attraction. The third term is randomization with α being the randomization parameter, and ϵ_i is a vector of random numbers drawn from a Gaussian distribution or uniform distribution. For e.g. the simplest form is ϵ_i and can be replaced by rand (1/2).

The algorithm presented here makes use of a synergic local search. Each member of the swarm explores the problem space taking into account results obtained by others, still applying its own randomized moves as well. The influence of other solutions is controlled by value of attractiveness. It can be adjusted by modifying two parameters: its maximum value β_0 and an

absorption coefficient γ. The first parameter describes attractiveness at $r_j = 0$ i.e. when two fireflies are found at the same point of search space S. In general $\beta_0 \in [0; 1]$ should be used.

Special cases:

For $\gamma \to 0$, the attractiveness is constant $\beta = \beta_0$, this is equivalent to say that the light intensity does not decrease in an idealized sky.

When $\gamma \to \infty$ leads to $\beta(r) \to \delta(r)$, which means that the attractiveness is almost zero in the sight of other fireflies.

i. Pseudo code:

1. Define objective function. F(x),

$X = (x_1, \ldots, x_d)^T$

2. Generate initial population of fireflies x_i ($i = 1, 2, \ldots, n$)

3. Determine light intensity I_i at x_i is determined by $f(x_i)$

Also define light absorption coefficient γ

4. Repeat until maximum generation is reached

For i = 1 to n= no. of fireflies

For j = 1 to n

If ($I_i < I_j$), move firefly I towards j; end if

Vary attractiveness with distance r via exp(-γr).

Evaluate new solutions and update light intensity. End for i

End for j

5. Rank the fireflies and find the current global best $\mathbf{g}_*$ goto step 4

6. end

ii. Advancements Application of FFA

Many bio-inspired algorithm exist that use simple rules and local information to create a global consensus of a single parameter, such as clock time in the flashing firefly algorithm, or location in the slime algorithm. In this paper we identify a control loop as the core structure of several bio-inspired algorithms. Two ways of combining these loops are identified, serially, and nested. These methods of combination can be used to combine algorithms to allow the control of multiple parameters (one per algorithm) on a global level without the use of a centralised point of control.

Thresholding is one of the most important techniques for performing image segmentation. It is generally simple and computationally efficient. The MEFFT algorithm simulates the behaviour of fireflies and the phenomenon of bioluminescent communication to develop the algorithm to select the adequate thresholds for image segmentation. The segmentation results of MEFFT algorithm are promising and it encourages further researches for applying this algorithm to complex and real-time image analysis problems such as target recognition, complex document analysis and biomedical image application [15]. The other application include synchronisation of wireless network which be seen in [16]. An application in pulse-coupled oscillator's model addresses the issue of synchronization of the oscillators with different frequencies. As a continuation of this research, it is important to study how the topology and commu1nication delay of the distributed oscillators system affect the synchronization process.

CHAPTER 3

COMPARATIVE ANALYSIS OF BFO, PSO AND FFA ON BENCHMARK PROBLEMS

3.1 BENCHMARK PROBLEMS:

In the field of evolutionary computation, it is common to compare different algorithms using a large test set, especially when the test involves function optimization. However, the effectiveness of an algorithm against another algorithm cannot be measured by the number of problems that it solves better. The ``no free lunch'' theorem shows that, if we compare two searching algorithms with all possible functions, the performance of any two algorithms will be, on average, the same. As a result of attempting to design a perfect test set where all the functions are present in order to determine whether an algorithm is better than another for every function, is a fruitless task.

That is the reason why, when an algorithm is evaluated, we must look for the kind of problems where its performance is good, in order to characterize the type of problems for which the algorithm is suitable. In this way, we have made a previous study of the functions to be optimized for constructing a test set with fewer functions and a better selection. This allows us to obtain conclusions of the performance of the algorithm depending on the type of function.

The test set has several well characterized functions that will allow us to obtain and generalize, as far as possible, the results regarding the kind of function involved. Nevertheless, we have added two functions to the test set with the aim of balancing the number of functions of each kind. These two new functions are the function of Rosenbrock extended to dimensions

and the function of Schwefel; both of them have been widely used in evaluative optimization literature.

A function is multimodal if it has two or more local optima. A function of variables is separable if it can be rewritten as a sum of functions of just one variable [Had64]. The separability is closely related to the concept of epistasis or interrelation among the variables of the function. In the field of evolutionary computation, the epistasis measures how much the contribution of a gene to the fitness of the individual depends on the values of other genes.

Non separable functions are more difficult to optimize as the accurate search direction depends on two or more genes. On the other hand, separable functions can be optimized for each variable in turn. The problem is even more difficult if the function is also multimodal. The search process must be able to avoid the regions around local minima in order to approximate, as far as possible, the global optimum. The most complex case appears when the local optima are randomly distributed in the search space.

The dimensionality of the search space is another important factor in the complexity of the problem. A study of the dimensionality problem and its features was carried out by Friedman

Sphere function has been used in the development of the theory of evolutionary strategies, and in the evaluation of genetic algorithms as part of the test set proposed by De Jong Sphere, or De Jong's function is a simple and strongly convex function. Schwefel's double sum function was proposed by Schwefel. Its main difficulty is that its gradient is not oriented along their axis due to the epistasis among their variables; in this way, the algorithms that use the gradient converge very slowly. Rosenbrock function, or De Jong's function is a two dimensional function with a deep valley with the shape of a parabola of the form that leads to

the global minimum. Due to the non-linearity of the valley, many algorithms converge slowly because they change the direction of the search repeatedly. The extended version of this function was proposed by Spedicato. Other versions have been proposed. It is considered by many authors as a challenge for any optimization algorithm. Its difficulty is mainly due to the non-linear interaction among its variables.

Here in this text comparison of the performances of algorithms in terms of various parameters w.r.t four benchmark functions have been taken as:

A. Sphere Function

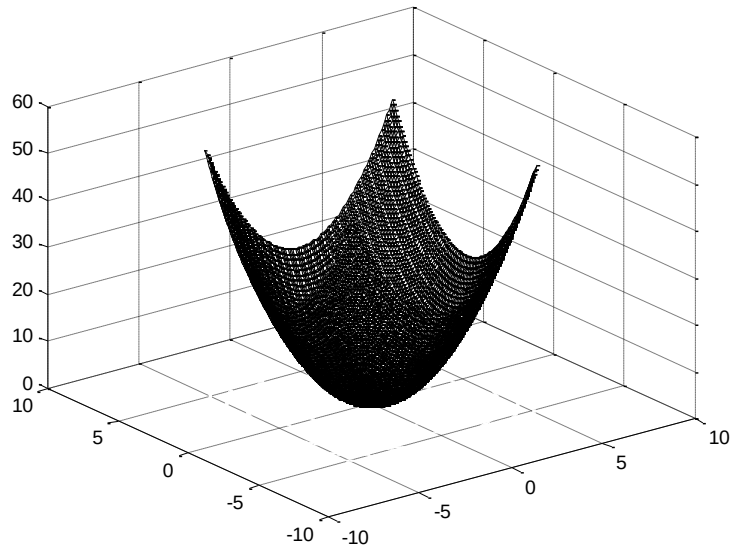


Fig. 3.1 Sphere function

$f(x, y) = x^2 + y^2$; -5.12 < x,y <5.12

Optimum point: 0 at (0, 0) (not a multimodal function but it is separable.)
The benchmark function mentioned above is 2-d spherical function and is unimodal and convex. It has global optimum point at 0.

B. Rosenbrock's function

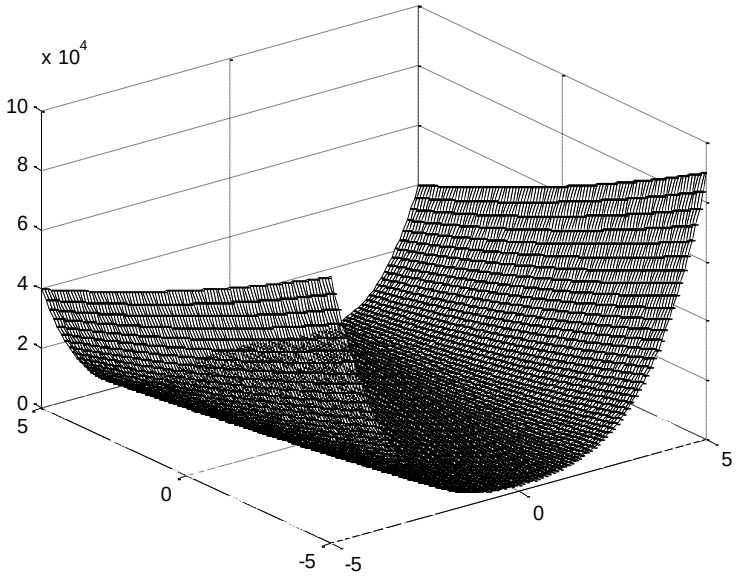


Fig. 3.2 Rosenbrock's function

$f(x,y) = (x-1)^2 + 100(y-x^2)^2 \; ; -5 < x,y < 5$

Optimum point 0 at (1,1).

This is 2-d rosenbrock's function having its global optima at $f_s = 0$ which occurs at (1, 1).

C. Michaelwicz's function

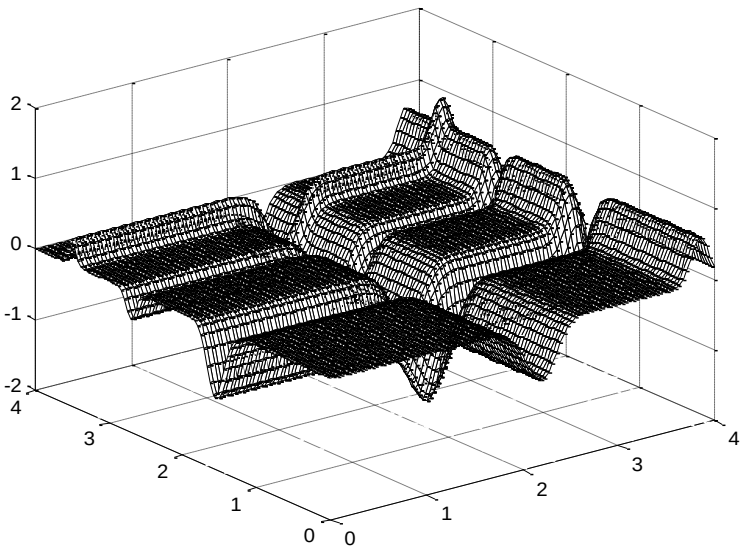


Fig. 3.3 Michaelwicz's function

f(x) = $\sin(x)sin^{20}\left(\frac{x^2}{\pi}\right) - \sin(y)\sin^{20}(\frac{2y^2}{\pi})$; 0 < x,y < 4

The global optimum of this function is -1.8013 at (2.20310, 1.57049).

D. Six hump camel back function.

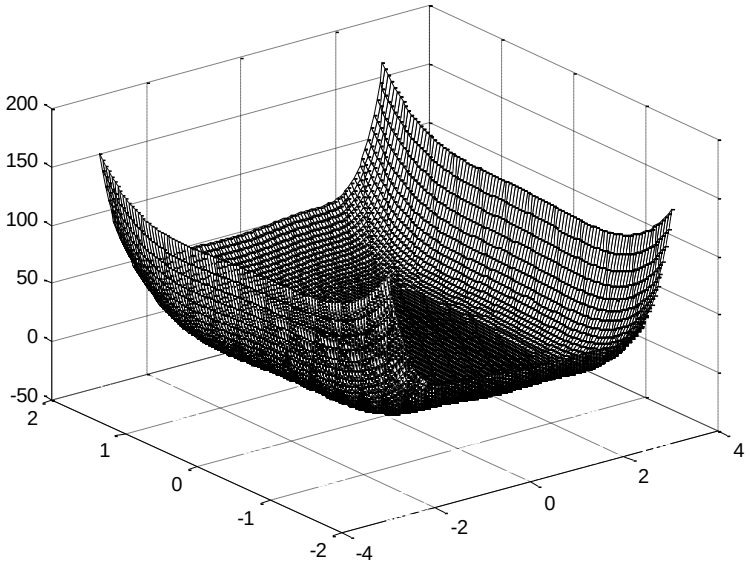


Fig. 3.4 Six hump camel back function

f(x,y) =($4 - 2.1x^2 + \frac{1}{3}x^4)x^2 + xy + 4(y^2 - 1)y^2$; -3 < x < 3, -2 < y < 2

This has global optima f_s =-1.0136 at (0.0808, -0.7120).

3.2 SIMULATION RESULTS OF BFO, PSO and FFA on BENCHMARK PROBLEMS

A. Benchmark function: Spherical

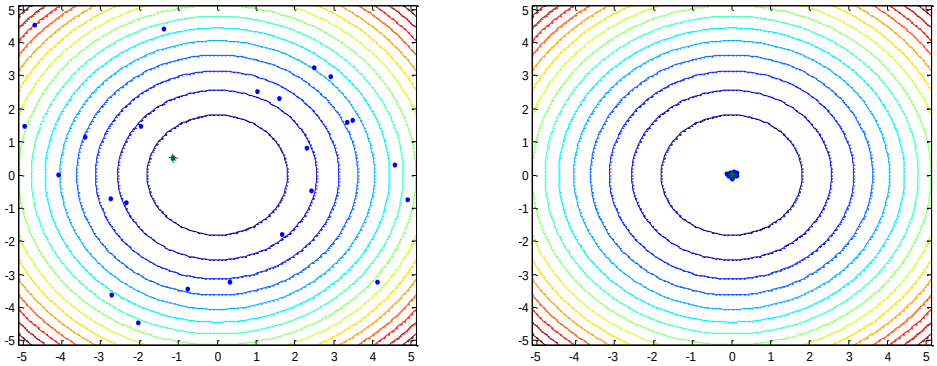


Fig.3.5 Initial and final positions of particles on Spherical function

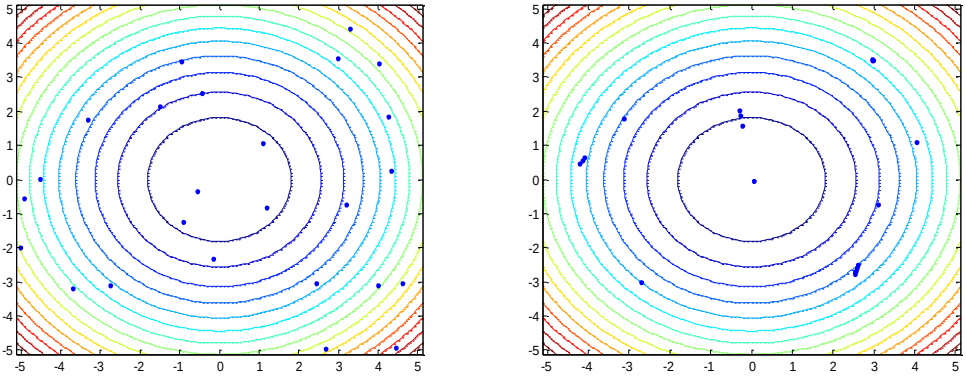


Fig.3.6 Initial and final positions of fireflies on Spherical function

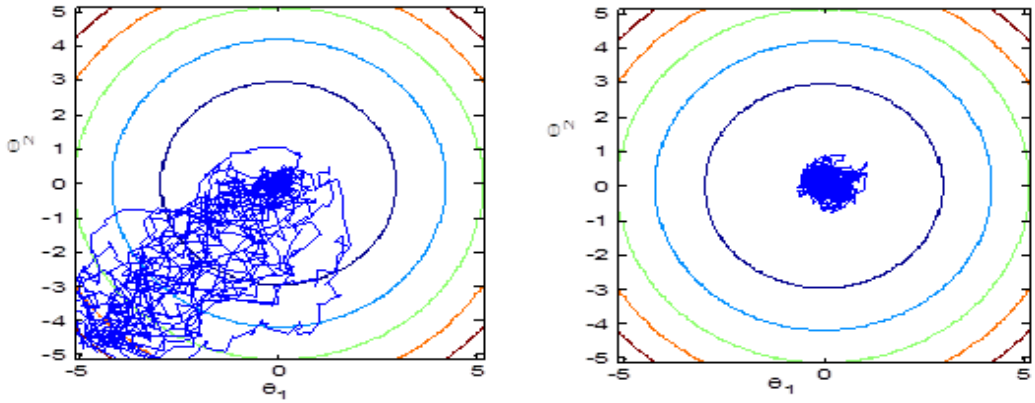


Fig.3.7 Initial and final positions of bacteria on Spherical function

Table 3.1 Performance results of algorithms on Sphere function in terms of elapsed time, mean and standard deviation

Algorithms	Elapsed time	Mean	Standard deviation

Firefly	2.589092 seconds	-17.4946	0
PSO	2.593764 seconds	-1.3165e-004	0
BFO	1.956597 seconds	-0.0294	0

The benchmark function mentioned above is 2-d spherical function and is unimodal and convex. It has global optimum point at 0. The best performing algorithm is analysed in terms of 3 attributes, elapsed time, mean (calculated as mean ($f(s^*)$ – min f(x,y))) and standard deviation (std($f(s^*)$ – min f(x,y))). BFO seems to have the best result among the three.

B. Benchmark function: Rosenbrock's function

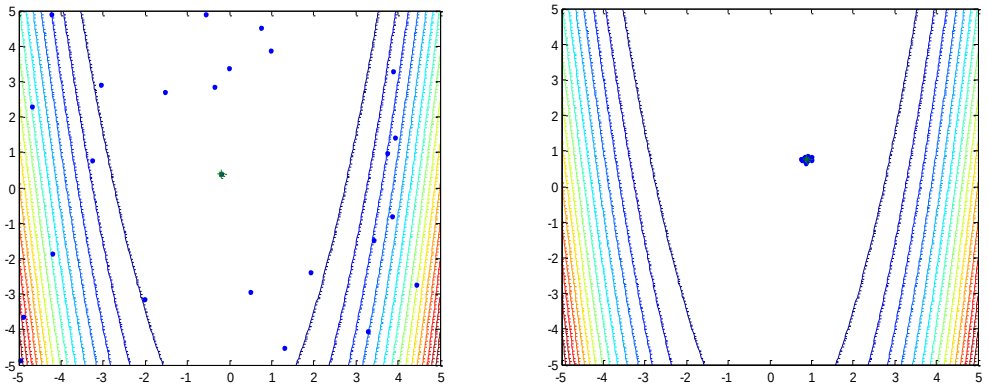


Fig.3.8 Initial and final positions of particles on Rosenbrock's function

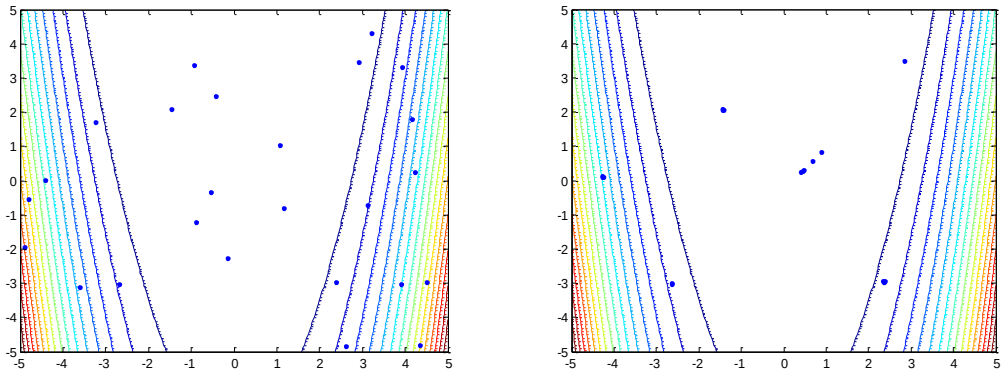


Fig.3.9 Initial and final positions of fireflies on Rosenbrock's function

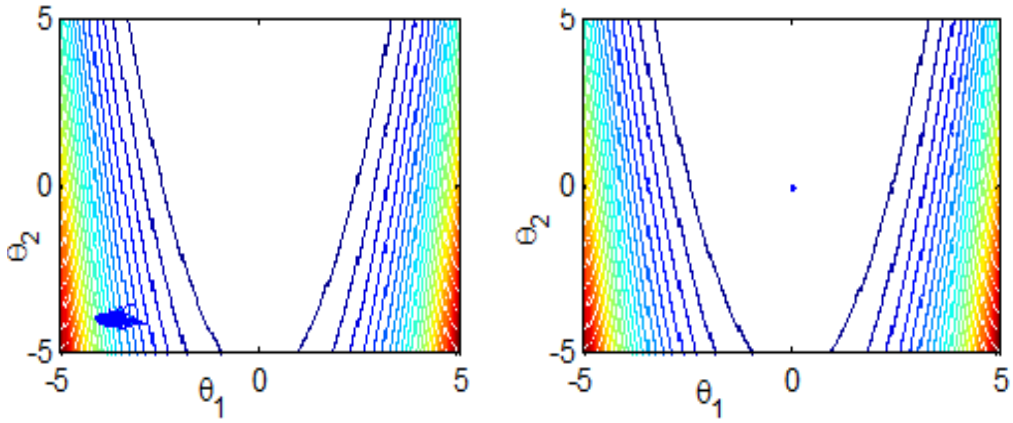


Fig.3.10 Initial and final positions of bacteria on Rosenbrock's function

TABLE 3.2: Performance of algorithms on Rosenbrock's function in terms of elapsed time, mean and standard deviation

Algorithms	Elapsed time	Mean	Standard deviation
Firefly	2.362486 seconds	-7.3886e+003	0
PSO	2.836760 seconds	-0.0159	0
BFO	1.439113 seconds	-51	0

This is 2-d rosenbrock's function having its global optima at $f_s = 0$ which occurs at (1,1). Here the best result seems to be that of particle swarm optimization.

C. *Benchmark function:* Michaelwicz's function

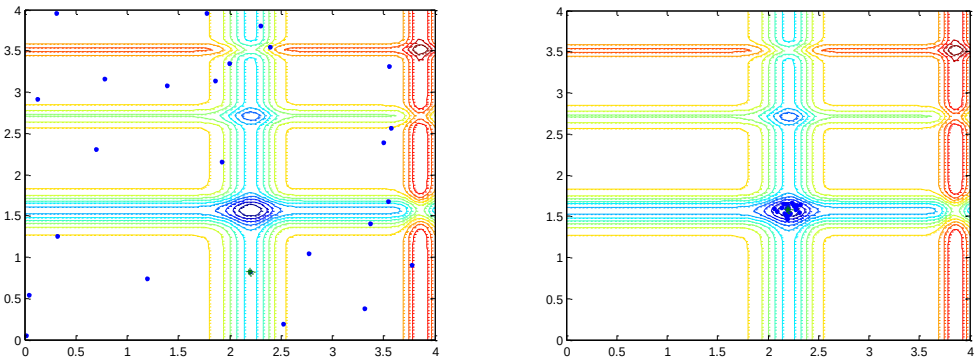


Fig.3.11 Initial and final positions of particles on Michaelwicz's function

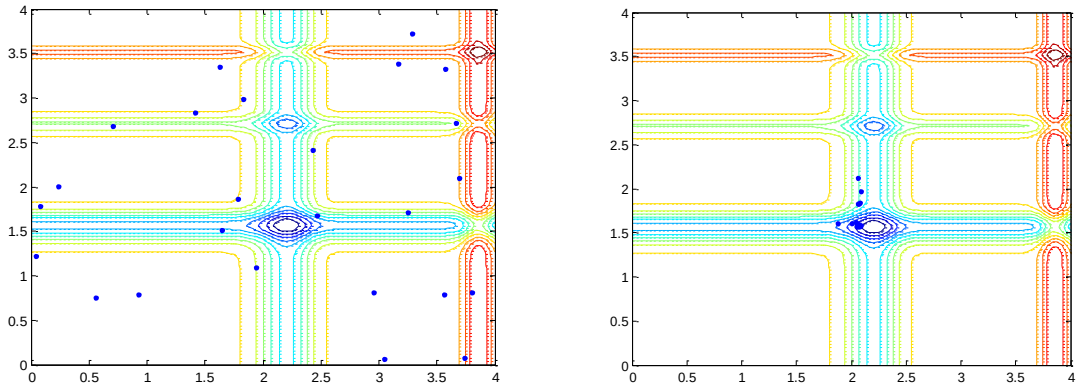


Fig.3.12 Initial and final positions of fireflies on Michaelwicz's function

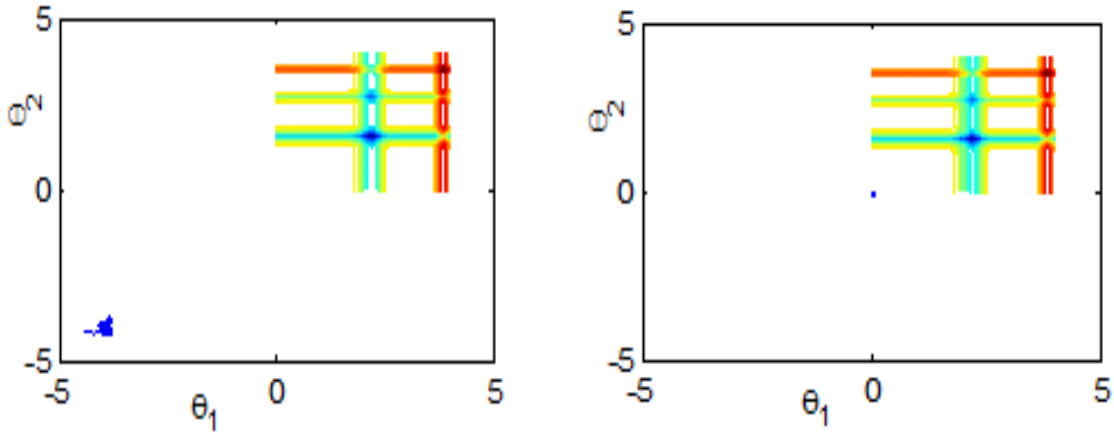


Fig.3.13 Initial and final positions of bacteria on Michaelwicz's function

Table 3.3: Performance results of algorithms on Michaelwicz's function in terms of elapsed time, mean and standard deviation

Algorithms	**Elapsed time**	**Mean**	**Standard deviation**
Firefly	2.685148 seconds	-0.7455	0
PSO	2.755020 seconds	-0.0013	0
BFO	1.483471 seconds	-1.8013	0

The global optima of this function are -1.8013 at (2.20310, 1.57049). Here again PSO performance better as compared to other two.

D. *Benchmark function:* Six hump camel back function

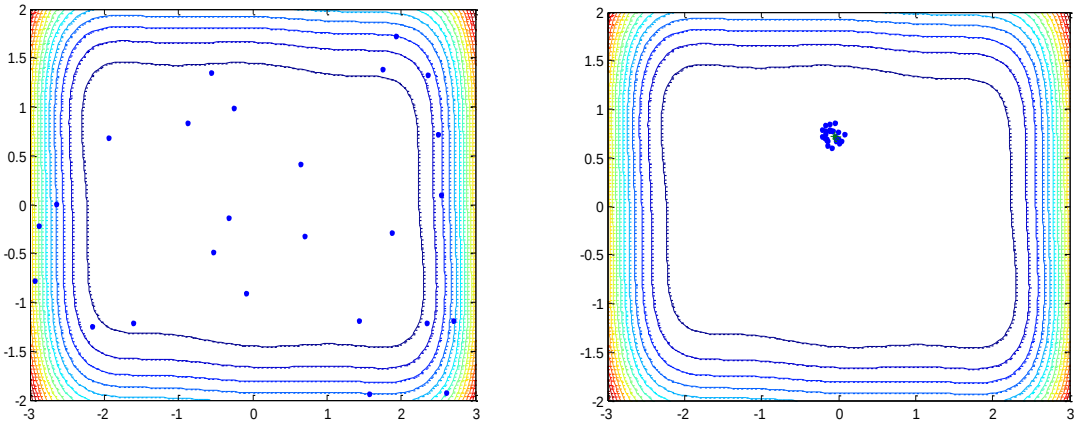


Fig.3.14 Initial and final positions of particles on Six hum camel back function

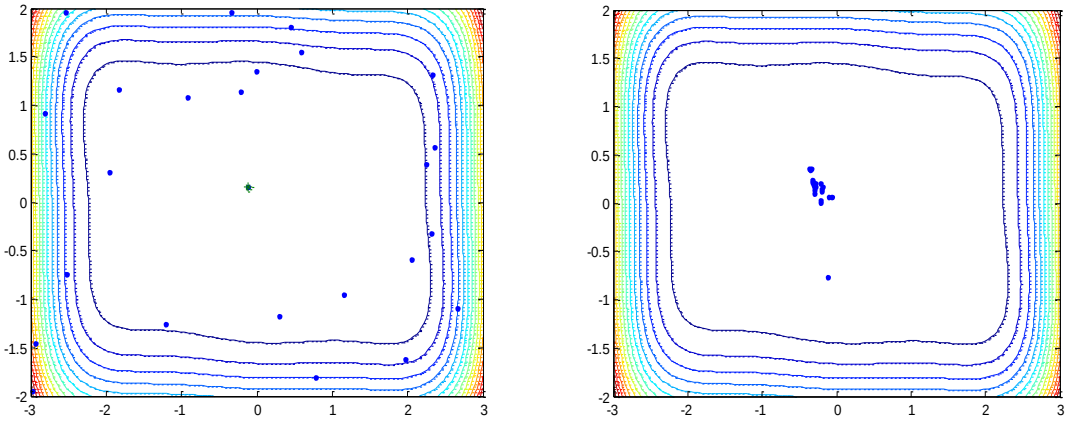


Fig.3.15 Initial and final positions of fireflies on Six hum camel back function

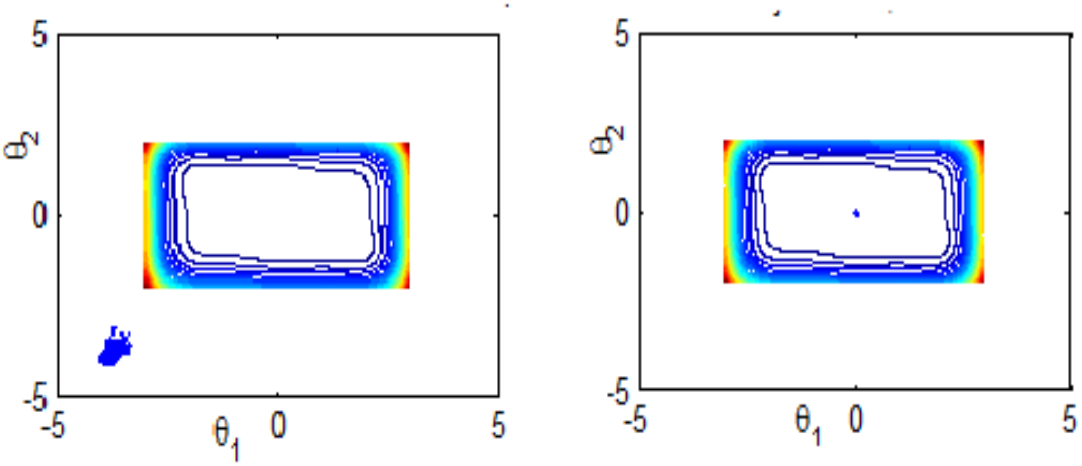


Fig.3.16 Initial and final positions of bacteria on Six hum camel back function

Table:3.4 Performance results of Algorithms on Six hump camel back function in terms of elapsed time, mean and standard deviation

Algorithm	Elapsed time	Mean	Standard deviation
Firefly	2.600843 seconds	-0.3604	0.2541
PSO	3.329348 seconds	0.0141	0
BFO	1.554232 seconds	-1.0136	0

This has global optima f_s =-1.0136 at (0.0808, -0.7120). Here again PSO has better convergence results and outperforms other two though has slightly more elapsed time.

Firefly is less stable in terms of standard deviation.

3.3 DISCUSSION ON RESULTS

All the algorithms have been conducted for same no. of population which is 25 and for 20 iterations. Based on this they have been compared on the basis of their elapsed time, mean and standard deviation to optimize the given benchmark function.

The firefly algorithm presented here makes use of a synergic local search. Each member of the swarm explores the problem space taking into account results obtained by others, still applying its own randomized moves as well. The influence of other solutions is controlled by value of attractiveness. It can be adjusted by modifying two parameters: its maximum value β_o and an absorption coefficient γ. The first parameter describes attractiveness at $r_j = 0$ i.e. when two fireflies are found at the same point of search space *S*. In general $\beta_0 \epsilon [0, 1]$ should be used and two limiting cases can be defined: when $\beta_0 = 0$, that is only non-cooperative distributed random search is applied and when $\beta_0 = 1$ which is equivalent to the scheme of cooperative local search with the brightest firefly strongly determining other fireflies positions, especially in its neighbourhood. On the other hand, the value of γ determines the variation of attractiveness with increasing distance from communicated firefly. Using $\gamma = 0$ corresponds to

no variation or constant attractiveness and conversely setting $\gamma \rightarrow 1$ results in attractiveness being close to zero which again is equivalent to the complete random search. In general γ ε [0,10] could be suggested. It is more convenient, however, to derive γ value specifically for the considered problem. Such customized absorption coefficient should be based on the characteristic length of the optimized search space. It is proposed here to use:

$$\gamma = \frac{\gamma_0}{r_{max}}$$

$$\gamma = \frac{\gamma_0}{r_{max}^2}$$

Where $\gamma \in [0,1]$ and:

$$r_{max} = max\, d(x_i, x_j);\ \forall\, x_i \& x_j \in S$$

The third parameter α can also be varied which here shows randomness. It is at α = 0.01 where we get best result of firefly algorithm.

Also for varied population in each algorithm the response varies. This not only increases complexity but takes larger amount of time. Bacterial foraging has least elapsed time for most of the benchmark functions. While that of firefly is maximum and also comes out to be least stable in terms of standard deviation. PSO proves to be the most efficient in terms of its convergence to the optimum point.

It is noticeable that Firefly is repeatedly outperformed by bacterial foraging algorithm and Particle Swarm Optimizer. It is also found that firefly is less stable in terms of standard deviation. Firefly Algorithm described here could be considered as an unconventional swarm-based heuristic algorithm for constrained optimization tasks. The algorithm constitutes a population-based iterative procedure with numerous agents (perceived as fireflies) concurrently solving a considered optimization problem. Agents communicate with each other via bioluminescent glowing which enables them to explore cost function space more effectively than in standard distributed random search. Most heuristic algorithms face the problem of

inconclusive parameters settings. Still the algorithm could benefit from additional research in the adaptive establishment of absorption coefficient and random step size. Furthermore some additional features like decreasing random step size and more sophisticated procedure of initial solution generation could bring further improvements in the algorithm performance. The algorithm could be hybridized together with other heuristic local search based technique like Adaptive Simulated Annealing. Thus by hybridising or providing certain modification to such less efficient algorithm can be made better performing algorithms.

3.4 CONCLUSION

Particle swarm optimization has shown the best results among the three taken algorithms. Though the results are not universal and may vary according to the objective function taken, yet it can be said that for the four functions taken with respect to each parameter PSO seems to have optimum results. Also it is noticeable that Firefly is repeatedly outperformed by bacterial foraging algorithm and Particle Swarm Optimizer. It is also found that firefly is less stable in terms of standard deviation. Firefly Algorithm described here could be considered as an unconventional swarm-based heuristic algorithm for constrained optimization tasks. The results of firefly can be improved by tuning several parameters along with slight modifications in it. This has been observed also while changing the value of absorption coefficient and its variation with the distance. Thus we can say that performance of any technique is objective dependent. There are various parameters that define the aptness of the algorithm so it depends on the requirements of the objective function and on the constraints imposed. For example if we take several benchmark problems and compare the results of three algorithms then it has been seen each would perform differently. Not only their individual performance vary even they tend to become less or more stable in terms of the parameters defining the quality of performance such as elapsed time, convergence rate, optimum value (maximised/minimized), mean and standard deviation etc. So depending upon our requirement we may define our objective function and the performance of algorithm for that particular function is based on how aptly the required objective is received, which could be in terms of final value, elapsed time, mean etc. Thus having witnessed these variations one can generalise the performance on overall terms and state any of the algorithm as best among the three taken.

CHAPTER 4

ADAPTIVE CONTROL USING BACTERIAL FORAGING ALGORITHM

4.1 ADAPTIVE CONTROL

Adaptive control is the control method used by a controller which must adapt to a controlled system with parameters which vary, or are initially uncertain. For example, as an aircraft flies, its mass will slowly decrease as a result of fuel consumption; a control law is needed that adapts itself to such changing conditions. Adaptive control is different from robust control in that it does not need *a priori* information about the bounds on these uncertain or time-varying parameters; robust control guarantees that if the changes are within given bounds the control law need not be changed, while adaptive control is concerned with control law changes themselves.

A. Classification of adaptive control techniques

In general one should distinguish between:

1. Feed forward Adaptive Control
2. Feedback Adaptive Control

There are several broad categories of feedback adaptive control (classification can vary):

- Dual Adaptive Controllers
  - Optimal Dual Controllers
  - Suboptimal Dual Controllers
- Non dual Adaptive Controllers
- Gain scheduling
- Model Reference Adaptive Controllers (MRACs)

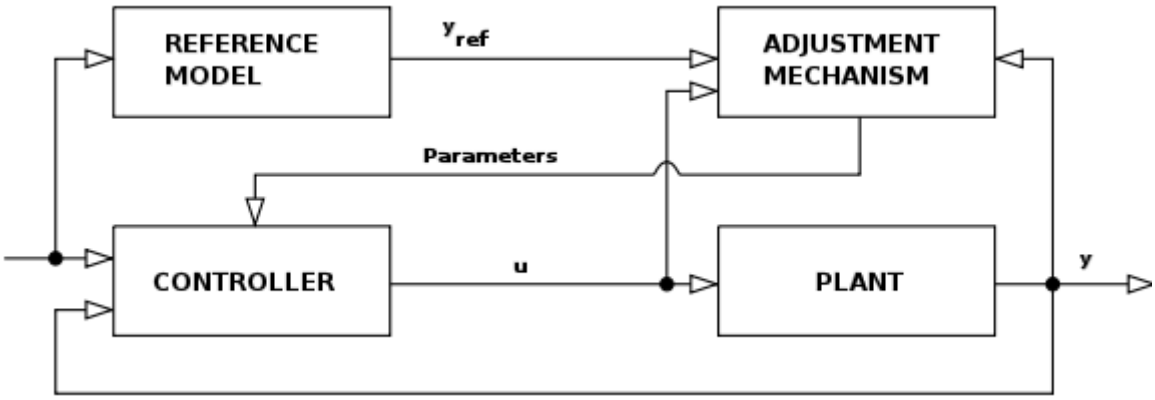


Fig.4.1 Model reference adaptive control

- Model Identification Adaptive Controllers (MIACs)

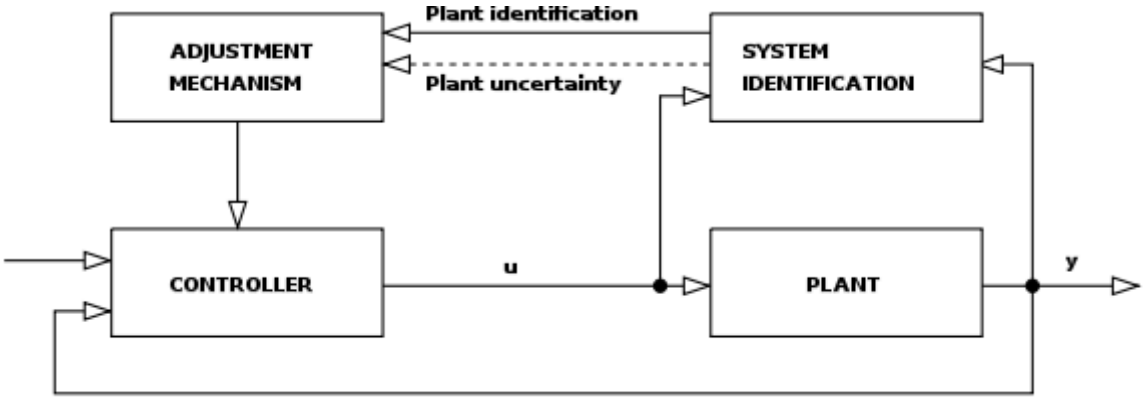


Fig.4.2 Model identification adaptive control (MIAC)

  - Cautious Adaptive Controllers [use current SI to modify control law, allowing for SI uncertainty]
  - Certainty Equivalent Adaptive Controllers [take current SI to be the true system, assume no uncertainty]
    - Nonparametric Adaptive Controllers
    - Parametric Adaptive Controllers
      - Explicit Parameter Adaptive Controllers
      - Implicit Parameter Adaptive Controllers

Some special topics in adaptive control can be introduced as well:

1. Adaptive Control Based on Discrete-Time Process Identification
2. Adaptive Control Based on the Model Reference Technique
3. Adaptive Control based on Continuous-Time Process Models
4. Adaptive Control of Multivariable Processes
5. Adaptive Control of Nonlinear Processes.

B. Strategies for adaptive control

i. Indirect adaptive control

There are at least two general approaches to adaptive control and in the first one we use online identification method to estimate the plant input-output mapping and a "controller designer" module to subsequently specify the parameters of the controller. Generally indirect adaptive controller can be taken as automating the mode-building and control design process that we use for fixed controller.

If the plant input-output mapping changes, the identifier will provide estimates of these changes and the controller design will subsequently time the controller. It is inherently assumed that we are certain that the estimated plant mapping is equivalent to the actual one at all times ("this is called certainity equivalence principle."). Then if controller designer can specify a controller for each set of plant parameter estimates, it will succeed controlling the plant. The overall approach is called "indirect adaptive control" since we tune the controller indirectly by first estimating the plant parameters

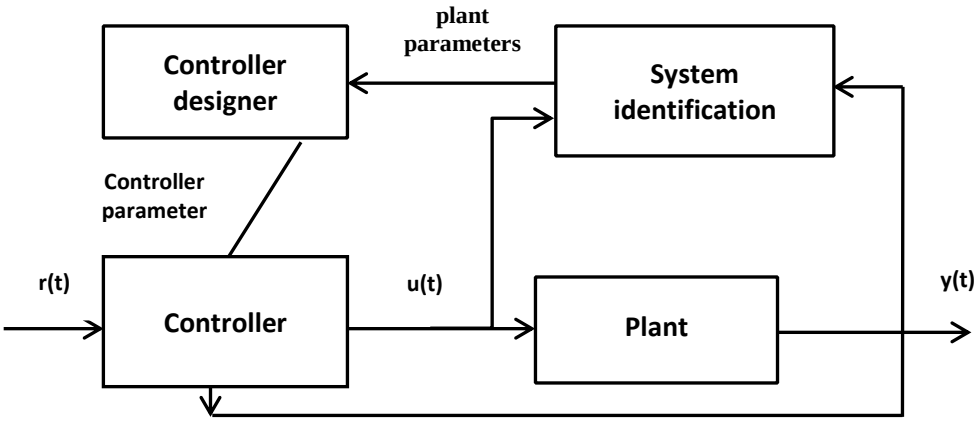


Fig.4.3 Indirect Adaptive Control

The structure used for the identifier model could be linear with adjustable coefficients. Alternatively, it could be a neural or fuzzy system with tuneable parameters that enter in a linear or non-linear fashion. Since the plant is assumed to be unknown, the non-linear mapping it implements is unknown. To do so, gradient or least square methods are used to tune neural or fuzzy systems for indirect adaptive control.

Alternatively, optimization method such as bio-mimicry of an individual foraging animal can be used.

ii. Direct adaptive control

Here "adaptation mechanism" observes the signals from the control system and adapts the parameters of the controller to maintain the performance even if there are changes in the plant. Sometimes, in either the direct or indirect adaptive controllers, the desired performance is characterised with a "reference model," and the controller then seeks to make the closed-loop called "model reference adaptive control" (MRAC).

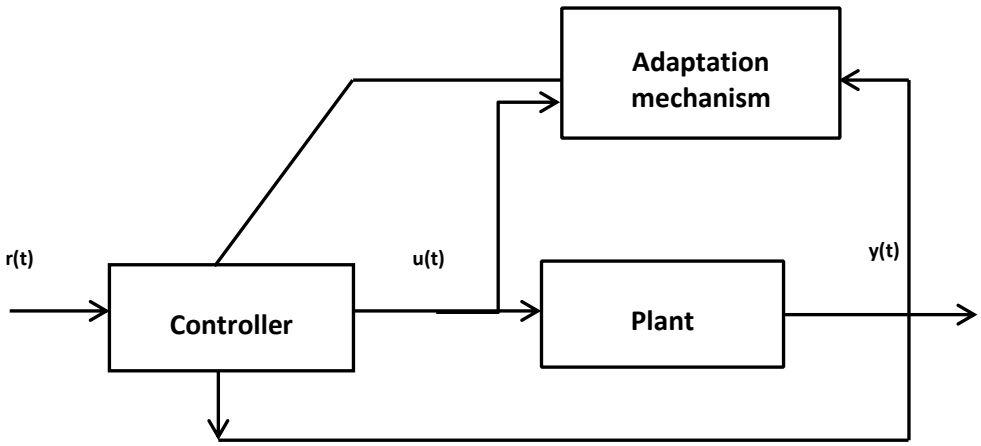


Fig.4.4 Direct adaptive control

In neural control or adaptive fuzzy control, the controller is implemented with neural or fuzzy system respectively. Normally, gradient or least squares methods are used to tune the controller. Also, many heuristic direct adaptive control methods have been developed, for instance based on reinforcement learning control.

Alternatively, bio mimicry method based on foraging, in this case, can be used to optimize and adjust controller parameter.

4.2 BACTERIAL FORAGING ALGORITHM BASED INDIRECT ADAPTIVE CONTROL

Here, foraging algorithm has been used as the basis for adaptive control. In indirect adaptive control one seeks to learn a plant model during the operation of a system. Learning is viewed as foraging for good model information (i.e. information that is truthful and useful for meeting goals). An identifier model is used which is a parameterized model of the plant, and consider foraging algorithm searching in the parameter space that corresponds to finding nutrients. Multiple identifier models and social foraging (i.e. multiple models are tuned simultaneously, with foragers possibly sharing information to try to improve foraging success). The BFA searches location in the parameter space, which corresponds to getting low identification errors between the model and the plant. Then, according to the sum of the squared identifier errors, at

each time instant the model that is the best and uses it in a standard certainty equivalence approach to specify a controller is chosen. Each identifier model is an affine mapping to match plant nonlinearities. The identifier model parameters represent the forager's position. The cost function for each forager, which defines the nutrient profile, is defined to be the sum of squares of past identifier error values for each identifier model. For parameter adjustment, a foraging algorithm is used that is based on E. coli chemotactic behaviour. Here a plant model is tuned in order to specify the controller parameters. A set (population) of approximators is used to tune and the optimization method used to tune the set is bacterial foraging optimization algorithm. Fig.4.5 shows the adaptive control using BFA.

`

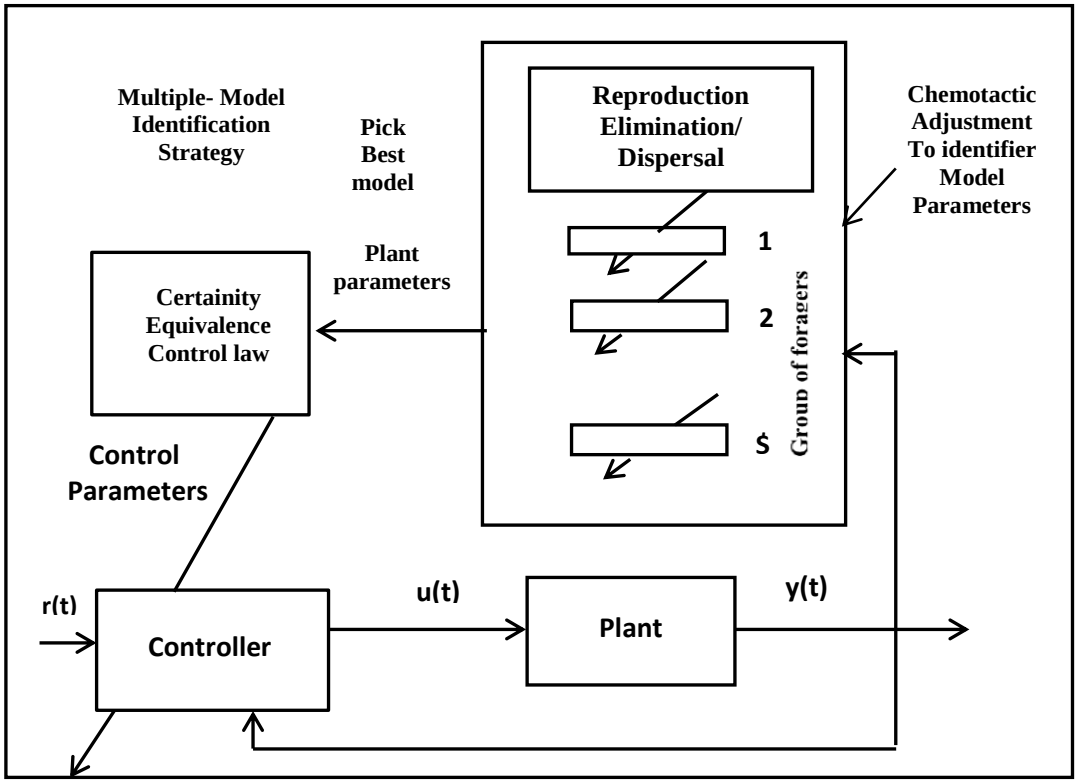


Fig.4.5 Adaptive control using BFA

Class of plant used is represented by

$$y(k+d) = \alpha\big(x(k)\big) + \beta\big(x(k)\big)u(k) \tag{4.1}$$

$\alpha(x(k))$ $and\ \beta(x(k))$ are unknown smooth functions of the state $x(k)$ while $y(k+d)$ is a nonlinear function of past values of u . $\beta(x(k)$ is requiring to be bounded away from zero.

$d \geq 1$ is the delay between the input and output. For d =1, $y(k+1) = \alpha(x(k)) + \beta(x(k))u(k)$ $y(k+d) = \alpha(x(k)) + \beta(x(k))u(k)$ (4.2)

$$y(k+d) = \alpha_u(x(k)) + \alpha_k(k) + (\beta_u(x(k)) + \beta_k(x(k))u(k) \quad (4.3)$$

Functions $\alpha_u(x(k))$ and $\beta_u(x(k))$ represent the unknown nonlinear dynamics of the plant. It is these functions which require to be estimated so that a controller can be specified. $\alpha_k(k)$ and $\beta_k x(k)$ are defined to be as known parts of the plant dynamics, these can be set to zero. $\beta(x(k))$ is assumed to satisfy

$0 < \beta_0 < \beta(x(k))$ for some known $\beta_0 > 0$ for all $x(k)$.

Estimation of an unknown ideal controller: An ideal controller is given by

$$u^*(k) = \frac{-\alpha(x(k))+r(k+d)}{\beta(x(k))} \quad (4.4)$$

This linearizes the dynamics of equation (15) such that $y(k) \to r(k)$. substituting $u(k) = u^*(k)$ in equation (15) we obtain $y(k+d) = r(k+d)$ so that tracking of reference input have been achieved within d steps.Since $\alpha(x(k))\ or\ \beta(x(k))$ are unknown, an estimator is developed for these plant nonlinearities and used them to form an approximation to $u^*(k)$. Using a "Certainty equivalence controller", the control input can be defined as

$$u(k) = \frac{-\tilde{\alpha}(x(k))+r(k+d)}{\tilde{\beta}(x(k))} \quad (4.5)$$

$\tilde{\alpha}(x(k))$ and $\tilde{\beta}(x(k))$ are estimates of $\alpha(x(k))$ and $\beta(x(k))$ respectively.

The certainty equivalence controller can be defined with the following estimates

$$\tilde{\alpha}\big(x(k)\big) = F_{\alpha}\big(x(k), \theta_{\alpha}(k)\big) \tag{4.6}$$

and $\tilde{\beta}\big(x(k)\big) = F_{\beta}\left(x(k), \theta_{\beta}(k)\right)$ (4.7)

Error $e(k) = r(k) - y(k)$ is not a linear function of the parameters.

Also

$e(k) = \tilde{y}(k) - y(k)$,

where $\tilde{y}(k)$ is estimate of (k) . A set of approximators for α and β where the i^{th} ones are denoted by

$F_{\alpha}\big(x, \theta_{\alpha}^{i}\big)$ and $F_{\beta}\big(x, \theta_{\beta}^{i}\big)$

for i=1,2,....,S.

From a foraging perspective, θ^{i} is viewed as the location of the i^{th} foragers in the environment. In foraging method position of the forager θ^{i} is used to minimize the fitness function $J(\theta^{i})$. Let the i^{th} estimate of the output and identification error be

$\tilde{y}^{i}(k+1) = F_{\alpha}\left(x(k), {\theta^{i}}_{\alpha}(k)\right) + F_{\beta}\left(x(k), {\theta^{i}}_{\beta}(k)\right)u(k)$ and $e^{i}(k) = \tilde{y}^{i}(k) - y(k)$ for i=1,2..., S. i^{th} Individual (bacteria) at time k can be given by

$\theta^{i}(k) = \left[{\theta^{i}}_{\alpha}^{T}(k), {\theta^{i}}_{\beta}^{T}(k)\right]$, i=1,2,..., S. (4.8)

Fitness function can be defined as

$$J\left(\theta^{i}(k-1)\right) = \left(e^{i}(k)\right)^{2}$$

$$= \left(\tilde{y}^{i}(k) - y(k)\right)^{2}$$

$$= \left(F_{\alpha}\left(x(k-1), \theta_{\alpha}^{i}(k-1)\right) + F_{\beta}\left(x(k-1), \theta_{\beta}^{i}(k-1)\right)u(k-1) - y(k)\right)^{2} \tag{4.9}$$

which measures the size of the estimation error for the i^{th} estimate. It is required to minimize $J\left(\theta^i(k-1)\right)$.

Forager's position in one dimension is given by θ_α and in the other dimension by θ_β so that forager's position is $\theta^i = [\theta_\alpha^i, \theta_\beta^i]^T, i = 1,2..,S$. S is the population size of the bacteria. Foraging strategy is based on E.coli chemotaxis, but without swarming, elimination-dispersal, and reproduction. Here chemotactic hill-climbing strategy is used to adjust the parameters. At each time step, one foraging step is used, that means either one tumble-tumble step. Foraging occurs while the control system operates with foraging (searching) for parameters occurring at each time step. For instance, if over one time step the cost did not decrease for an individual, then there is a tumble, and by this, a random direction is generated which update the parameters (location of the forager) in that direction. If, cost is improved from the last step, then another step in the same direction taken last time is made. In such case, forager is on a run in a good direction, down the cost function.

4.3 DYNAMICS OF LIQUID LEVEL SYSTEM

***Design Problem*:** In this problem we will study the development of indirect adaptive controllers for the liquid level process control problem. From the foraging perspective, we view θ^i as the location of the i^{th} forager in its environment. In a foraging method, we will move the position of the forager θ^i so as to minimize J (θ^i). The particular manner used to adjust the θ^i (k-1) to find θ^i(k) will depend on the choice of the foraging algorithm steps. In an indirect adaptive control strategy, we view foraging as searching for good model information. If a foraging strategy is used, we view θ^i (k) as the forager who has found the best model information. With foraging strategy, we could view the fixed- position members, "information

centres", if foragers were endowed with communication capabilities. If such centres have good model information they will tend to attract foragers.

Planning for a process control problem in this section we will develop a planning strategy ofr every simple yet representative process control system and then we design and test a planning strategy.

***Level Control in Surge Tank*:** Consider the "Surge tank", shown in figure that can be modelled by

$$\frac{dh(t)}{dt} = -\frac{\bar{d}\sqrt{2gh}}{A(h(t))} + \frac{\bar{c}}{A(h(t))}u(t) \qquad (4.10)$$

Where u(t) is the input flow (control input), which can be positive or negative (it can both pull liquid out of the tank and put in it), h(t) is the liquid level (the output of the plant);

$A\,h(t) = |\bar{a}h(t) + \bar{b}|$ is the cross sectional area of the tank and $\bar{a} > 0$ and $\bar{b} > 0$; g = 9.8 ;

$c \in [0.9, 1]$ is a "clogging factor" for a filter in pump actuator where if $c = 0.9$, there is some clogging of filter and if c = 1, the filter is clean so there is no clogging ; and $d > 0$, parameter is related to diameter of the output pipe. We think of all these plant parameters as being fixed for a particular surge tank; however, we could consider other values for these parameters and test the controller for these.

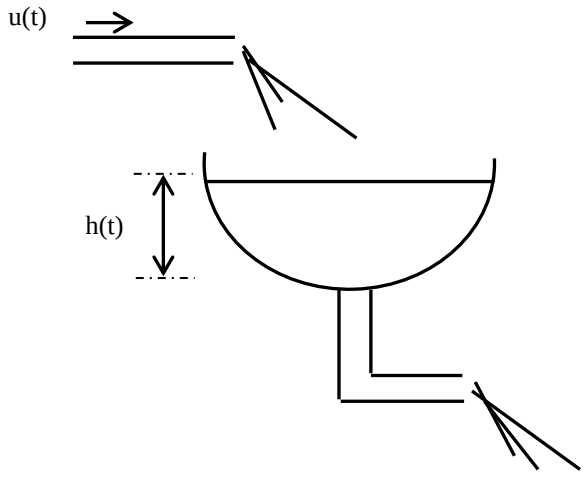


Fig. 4.6 Liquid level System

Let r(t) be desired level of the liquid(reference i/p) and e(t) = r(t) – h(t) be the tracking error. Assume the knowledge of reference trajectory and priori and assumes that r(t) ϵ [0.1, 8] and that we will not have h(t) > 10, Assume that h(0) = 1.

To convert to a discrete time approach we use Euler's approximations to the continuous dynamics to obtain

$$h(k+1) = h(k) + T\left[\frac{-\bar{d}\sqrt{19.6h(k)}}{|\bar{a}h(k)+b|} + \frac{\bar{c}}{|\bar{a}h(k)+b|}u(k)\right] \quad (4.11)$$

where T = 0.1. We assume that the plant input saturates at ± 50 so that if the controller generates on i/p $\bar{u}$(k), then

$$\begin{aligned} u(k) &= 50 \; if \; \bar{u}(k) > 50 \\ &= \bar{u}(k) \; if \; -50 \leq \bar{u}(k) \leq 50 \\ &= \bar{u}(k) < -50 \end{aligned} \quad (4.12)$$

Also, to ensure that the liquid level never goes negative, we simulate our plant using

$$h(k+1) = \max\{0.001, h(k) + T\left[\frac{-\bar{d}\sqrt{19.6h(k)}}{|\bar{a}h(k)+b|} + \frac{\bar{c}}{|\bar{a}h(k)+b|}u(k)\right]\} \quad (4.13)$$

A. *Simulation results and discussion on results*

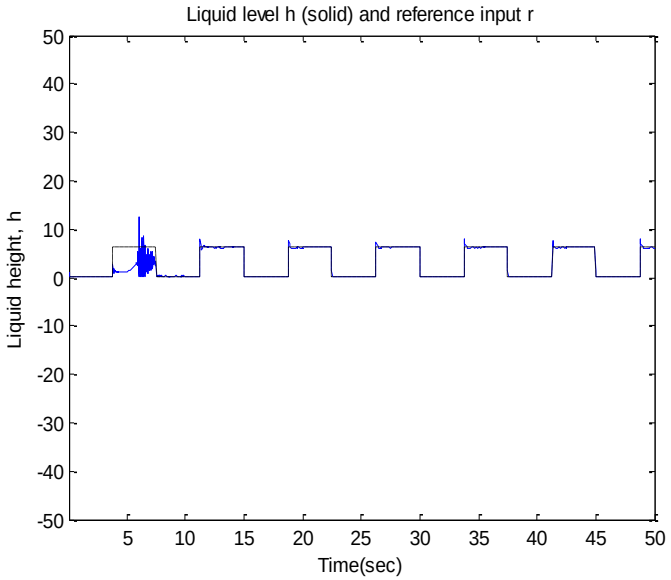


Fig.4.7 Liquid height vs. reference input

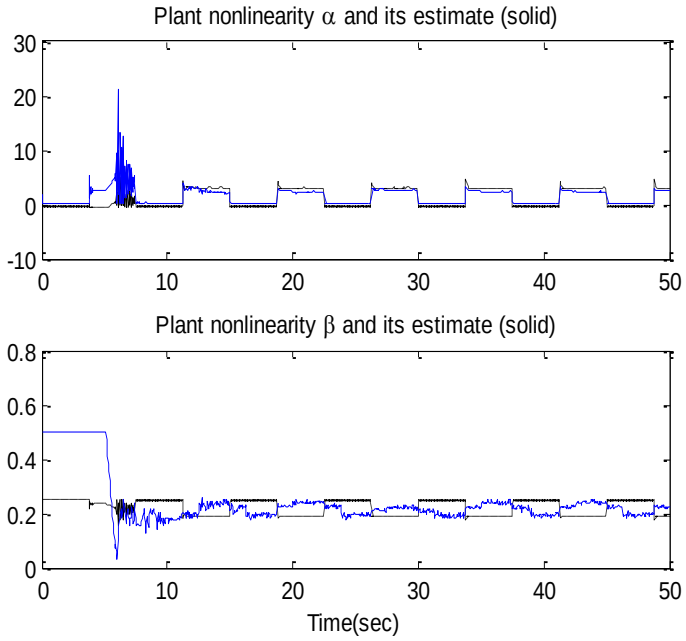


Fig. 4.8 Plant non-linearities α and β and their estimates

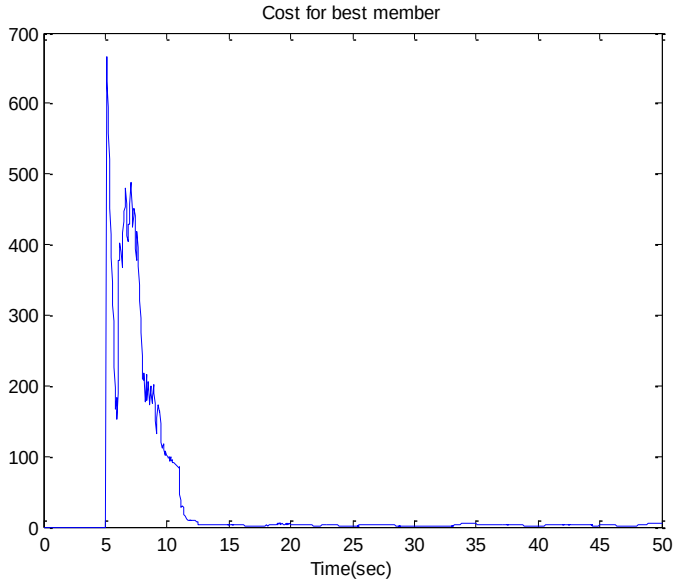


Fig. 4.9 Cost of the best member of population

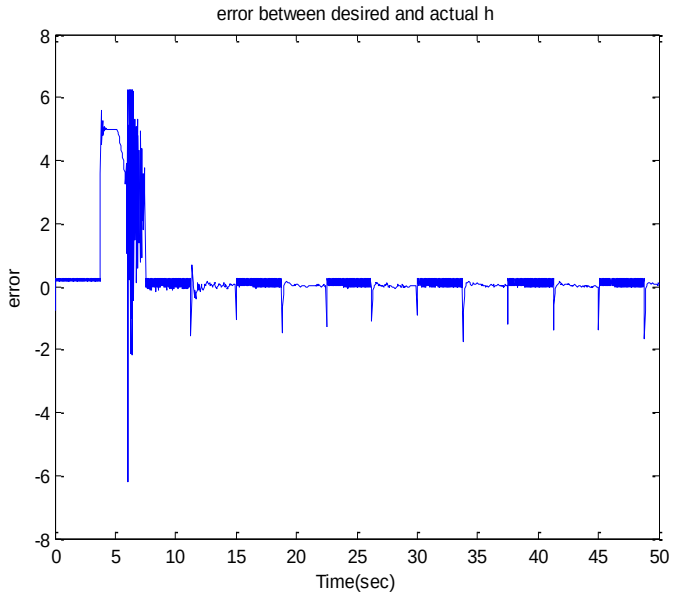


Fig. 4.10 Error between desired and actual height

Our forager's position in one dimension is given by θ_α and in the other dimension by θ_β so our i^{th} forager's position is $\theta^i = [\theta_\alpha{}^i \ \theta_\beta{}^i]^T$, i=1,2,3,…,S . We choose S= 600 as the population size of bacteria. Foraging strategy employed is based on e.coli chemotaxis, but without swarming, elimination-dispersal, and reproduction. Hence, we only use the chemotactic hill-climbing strategy to adjust the parameters. At each time step, we take one foraging step, which for our foraging strategy means that we use either one tumble-tumble step or part of a "run". In

this way, the foraging occurs while the control system operates with foraging (searching) for parameters occurring at each time step. For instance, if over one time step the cost did not decrease for an individual, then there is a tumble, and to do this, we generate a random direction and update the parameters in that direction. If, however the cost improved from the last step, then another step is taken in the same direction, as taken in last step. In this case the forager is on a "run" in a good direction, down the cost function.

The step size C (i,k) is set to be 0.05 for all bacteria for all times. The maximum number of steps along a good direction is $N_s = 4$ and $\theta^i_\alpha(1) = 2$, $\theta^i_\beta(1) = 0.5$, i=1, 2, 3… S. We use the cost evaluation procedure with $J_s(\theta^i(k-1),N)$ with N=100, with chemotactic steps = 800.

The performance of closed loop system is illustrated in fig.4.7 where we see that, after an initial transient period that results in part due to the poor initialization of the estimators and the controller's start-up method. To further illustrate some properties of adaptive controller we plot the cost of best individual in the population and the index i of the best individual in population for every time step.

Firstly, note that early in simulation cost is zero due to how we start up the controller. Then we start the controller at t=5 sec the cost jumps to a relatively high value, this represents that we have a poor initialization for the population. After some time, however, the foraging strategy is somewhat successful at adjusting the population members so that the estimation error decreases and hence, the beet cost decreases. Note, however that cost does not always decrease over time. It can also increase and one cause of this can be the change in the reference input.

4.4. DYNAMICS OF DC SERVOMOTOR

***Design problem*:** In this problem we will study the development of indirect adaptive controllers for the DC Servomotor. Here we intend to use foraging perspective to follow the

desired trajectory by dc servomotor by tuning the parameters and using the chemotaxis step of BFO. Reproduction and elimination-dispersal are not used here.

A high performance drive system consists of a motor and a controller integrated to perform a precise mechanical maneuverer. This requires the shaft speed and/or position of the motor to clearly follow a specified trajectory regardless of unknown load variations and other parameter uncertainties.

BFO is used to emulate the unknown nonlinear plant dynamics by presenting a suitable set of input/output patterns generated by the plant. Once system dynamics have been identified using a BFO conventional control techniques can be applied to achieve the desired objective trajectory tracking.

DC Motor model

The DC motor dynamics are given by the following equations

$$v_a(t) = R_a i_a(t) + L_a \frac{di_a}{dt} + e_b(t) \quad (4.14)$$

$$e_b(t) = K_b w(t) \quad (4.15)$$

$$T_M(t) = K_T i_a(t) \quad (4.16)$$

$$= J\frac{dw(t)}{dt} + Bw(t) + T_L(t) + T_F \quad (4.17)$$

where

$v_a(t)$ = applied a armature voltage (volts);

e_b = back emf (volts);

$i_a(t)$ = armature current(amps);

R_a = armature winding resistance (ohms);

L_a = armature winding inductance (henrys);

$W(t)$ = armature velocity of the motor rotor (rad/sec);

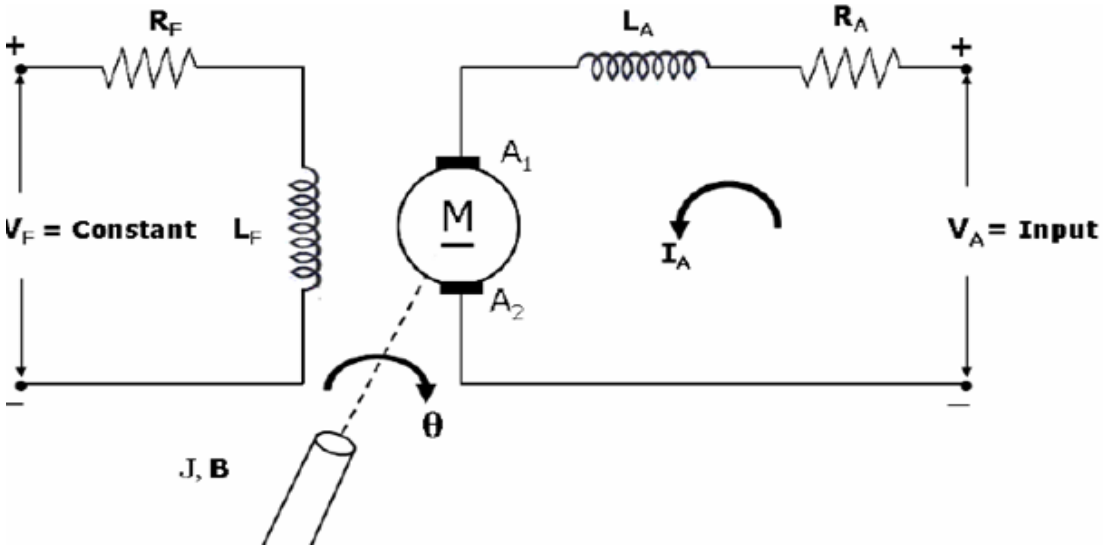


Fig. 4.11 DC Servomotor

T_M = Torque developed by Motor in N-m

K_t = Torque constant in N-m/A

J = Moment of inertia of Motor in Kg-m2/ rad

B = Frictional Coefficient of motor N-m/ (rad/sec)

K_b = Back emf Constant in Volt / (rad/sec)

$T_L(t)$ = Disturbance Load torque(newton-m).

The load torque $T_L(t)$ can be expressed as

$$T_L(t) = \psi(w) \tag{4.18}$$

Where $\psi(.)$ depends on the nature of the load.

For the most propeller driven system or fan type loads, the function $\psi(.)$ takes the following form

$$T_L(t) = \mu\psi^2(t)[\text{sgn } w(t))]$$

Where μ is constant.

DC motor drive system can be expressed as single-input, single-output, system by combining equations (4.4) and (4.8):

$$L_aJ\frac{d^2w(t)}{dt^2} + (R_aJ+L_aB)\frac{dw(t)}{dt} + (R_aB + K_bK_T)w(t) + L_a\frac{dT_L(t)}{dt} + R_a[T_L(t) + T_F] +$$

$$K_T v_a(t)=0 \tag{4.19}$$

The discrete-time model is derived by replacing all continuous diffrentials with finite diffrences.

$L_aJ[\frac{w(k+1)-2Iw(k)+w(k-1)}{T^2}]$ + $(R_aJ + L_aB)[\frac{w(k+1)-w(k)}{T}]$ + $(R_aB + K_bK_T)w(k)$ + $L_a[\frac{T_L(k)-T_L(k-1)}{T}]$ + $R_aT_L(k) + R_aT_F + K_Tv_a(k) = 0$ (4.20)

$T_L(k) = \mu w^2(k)[\text{sgn } w(k)]$ (4.21)

$T_L(k\text{-}1) = \mu w^2(k\text{-}1)[\text{sgn } w(k)]$ (4.22)

T = sampling period

$w(k) = w(t=kT)$; k = 0,1,2,…

manipulating 4.11-4.13 yields

$w(k+1)= K_1w(k) + K_2w(k\text{-}1) + K_3[\text{sgn } w(k)]\, w^2(k) + K_4[\text{sgn } w(k)]w^2(k\text{-}1) + K_5v_a(k) + K_6$

J = 0.068 kg-m^2

B = 0.03475N-m(rad/sec)

$R_a = 7.56\Omega$

L_a = 0.055H

K_T = 3.475 N-m/amp

K_b = 3.475 volts/(rad/sec)

M = 0.0039 N-m/$(rad/sec)^2$

T_F = 0.212 N=m

T = 40 msec= 0.04 sec

With these parameters, the constants K_1, K_2, K_3, K_4, K_5 and K_6 become

K_1 = 0.34366

K_2 = -0.1534069

K_3 = -2.286928 x 10^{-3}

K_4 = 3.5193358K_5 x 10^{-4}

K_5 = 0.2280595

K_6 = -0.105184

A. Simulation results and discussion on results

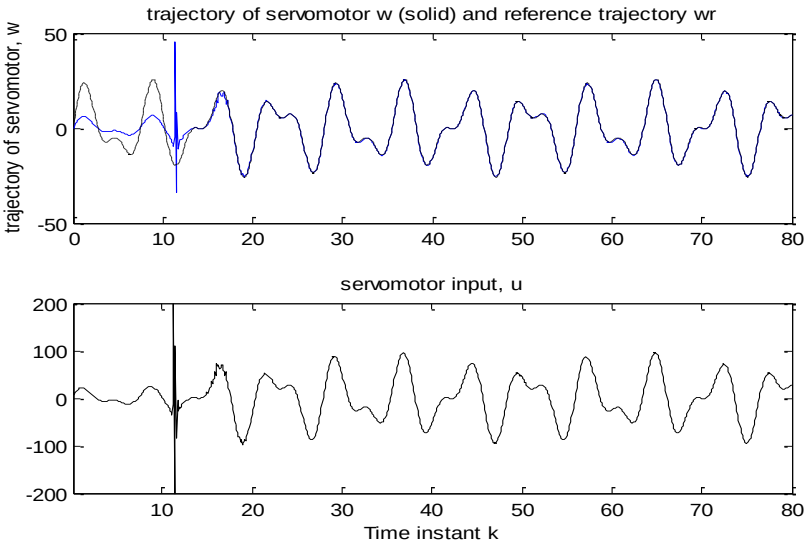


Fig. 4.12 Trajectory of Servomotor vs. Reference trajectory

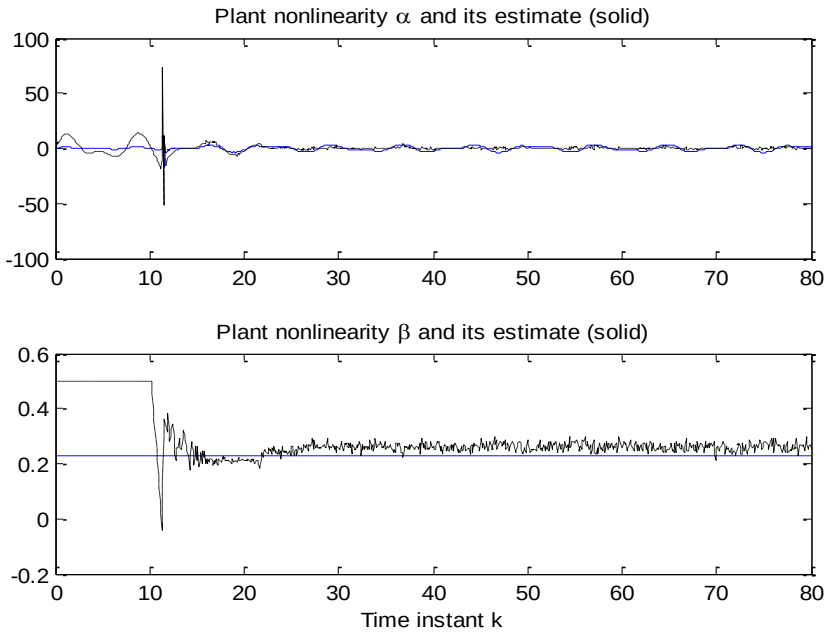


Fig. 4.13 Estimation of Non-linearities α and β of plant

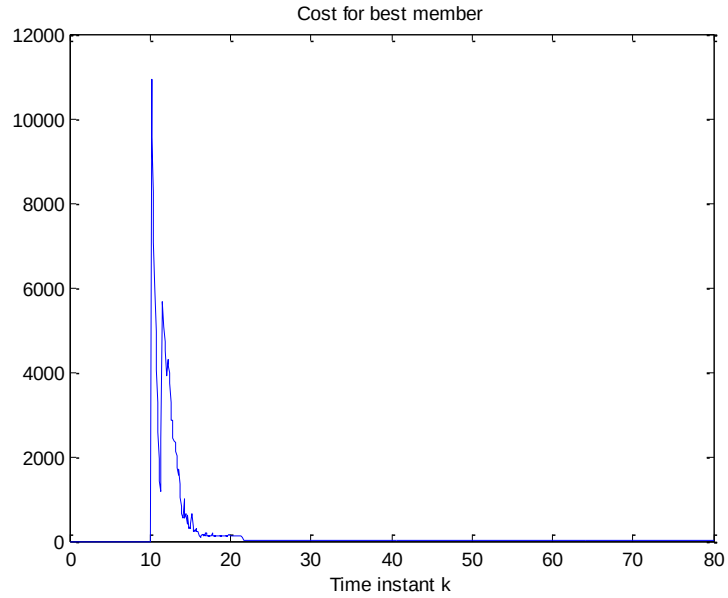


Fig. 4.14 Cost of best member w.r.t time

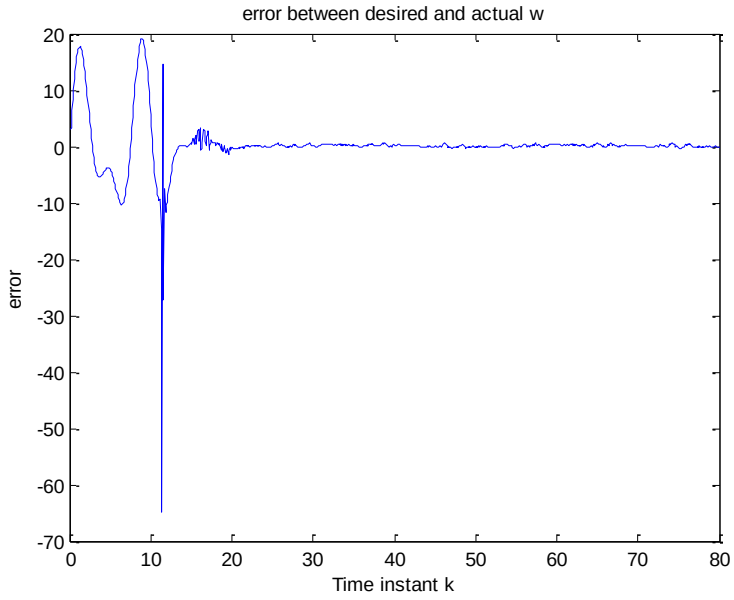


Fig. 4.15 Error between desired and actual output

Our forager's position in one dimension is given by θ_α and in the other dimension by θ_β so our i^{th} forager's position is $\theta^i = [\theta_\alpha{}^i \ \theta_\beta{}^i]^T$, i=1,2,3,…,S. We choose S= 1000 as the population size of bacteria. Foraging strategy employed is based on e.coli chemotaxis, but without swarming, elimination-dispersal, and reproduction. Hence, we only use the chemotactic hill-climbing strategy to adjust the parameters. At each time step, we take one foraging step, which for our

foraging strategy means that we use either one tumble-tumble step or part of a "run". In this way, the foraging occurs while the control system operates with foraging (searching) for parameters occurring at each time step. For instance, if over one time step the cost did not decrease for an individual, then there is a tumble, and to do this, we generate a random direction and update the parameters in that direction. If, however the cost improved from the last step, then another step is taken in the same direction, as taken in last step. In this case the forager is on a "run" in a good direction, down the cost function.

The step size C (i, k) is set to be 0.05 for all bacteria for all times. The maximum number of steps along a good direction is $N_s = 4$ and $\theta^i_\alpha(1) = 2.1$, $\theta^i_\beta(1) = 0.5$, i=1, 2, 3,…, S. We use the cost evaluation procedure with J_s $(\theta^i(k-1), N)$ with N=100, with chemotactic steps = 800.

The performance of closed loop system is illustrated in fig. 4.12 where we see that, after an initial transient period that results in part due to the poor initialization of the estimators and the controller's start-up method. To further illustrate some properties of adaptive controller we plot the cost of best individual in the population and the index i of the best individual in population for every time step.

Firstly, note that early in simulation cost is zero due to how we start up the controller. Then we start the controller at t = 10 sec the cost jumps to a relatively high value, this represents that we have a poor initialization for the population. After some time, however, the foraging strategy is somewhat successful at adjusting the population members so that the estimation error decreases and hence, the beet cost decreases. Note, however that cost does not always decrease over time. It can also increase and one cause of this can be the change in the reference input.

CHAPTER 5

CONCLUSION

Study of evolutionary algorithm to bio-inspired algorithms has shown how optimization has become an important area of research in every field of engineering. BFA, PSO and FFA have been studied in detail along with their application areas, through research papers, published in several streams of soft computing, controls and image processing etc.

Analysis of Algorithms has thrown light on several aspects of optimization and has also given the direction to carry the work ahead. It has been observed that the effectiveness of an algorithm against another algorithm cannot be measured by the number of problems that it solves better. The ``no free lunch" theorem shows that, if we compare two searching algorithms with all possible functions, the performance of any two algorithms will be, on average, the same. As a result of attempting to design a perfect test set where all the functions are present in order to determine whether an algorithm is better than another for every function, is a fruitless task.

But depending upon the priority and constraints as imposed on the problem we can find the best algorithm in terms of the parameters that are important for any specified problem. Thus the same has been observed with respect to three parameters, elapsed time, mean and standard deviation for the three algorithms on four benchmark problems. The best as per these parameters was found to be particle swarm optimization and even bacterial foraging algorithm had equivalent performance

BFA has been taken further to deal with indirect adaptive control problems, where in only one basic step of the algorithm (Chemotaxis) to choose the best member among controllers is used. Non-linearities of plant have been estimated and trajectory followed by the

system is also observed. Results obtained are satisfactory and can be further improved by the changing the parameters of system and algorithms as well (such as chemotactic steps, step size, population etc.) Other techniques can also be used to do the same and depending upon the objective function modifications can be made in the algorithms as performance of any technique is objective dependent. There are various parameters that define the aptness of the algorithm so it depends on the requirements of the objective function and on the constraints imposed.

FUTURE SCOPE AND AREA OF RESEARCH: Various other control applications could also be implemented using these algorithms. Depending upon the application Algorithm can be modified and its parameters could be tuned to have better results. Algorithm like Harmony search, Simulated Annealing, Bee Algorithm and Firefly Algorithm can be further studied and applied to these applications.

REFERENCES

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," *Proc.of IEEE Int. Conf. Neural Networks*, vol. 4, pp. 1942–1948, 1995.

[2] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory." *Proc.of 6th Int. Symp. Micro Machine and Human Science (MHS '95)*, pp. 39–43, 1995.

[3] B. D. Hughes, *Random Walks and Random Environments*. London, U.K.: Oxford Univ. Press, 1996.

[4] J. R. Koza, F. H Bennett 111, D. Andre, and M. A. Keane, "Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming." *Proc. of the First Annual Conference at Cambridge, MA,* pp. 123-131, 1996. The MIT Press.

[5] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer. "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems." *Trans of IEEE on Evolutionary Computation,* 1997.

[6] R. W. Morrison and K. A. De Jong, "A test problem generator for non-stationary environments", *Proc. of IEEE Congress on Evolutionary Computation*, IEEE Press, pp. 2047–2053, 1999.

[7] WANG Ling, "Intelligent Optimization Algorithms with Applications." Beijing: Tsinghua University Press, 2001.

[8] K. M. Passino, "Bio mimicry of bacterial foraging for distributed optimization and control," *IEEE Control Syst.*, vol. 22, no. 3, pp. 52–67, Jun. 2002.

[9] Daniel W. Boeringer and Douglas H. Werner, "Particle Swarm Optimization versus Genetic Algorithms for Phased Array Synthesis." *Trans*. Of *IEEE on Antennas and Propagation, vol. 52, no. 3*, pp. 771-779, Mar. 2004.

[10] F. Van den Bergh and A.P. Engelbrecht , "A Cooperative Approach to Particle Swarm Optimization" , *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225—239 , June 2004.

[11] PY. Yin, "A discrete particle swarm algorithm for optimal polygonal approximation of digital curves," *J. Vis. Commun. Image R.* vol. 15, pp. 241-260, 2004.

[12] J. J. Liang and A. K. Qin, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions." *Trans of IEEE on Evolutionary Computation,* vol. 10, no. 3, Jun., 2006.

[13] W. J. Tang and Q. H. Wu, "Bacterial Foraging Algorithm For Dynamic Environments." *Proc. of IEEE Congress on Evolutionary Computation*, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada, pp. 1324-1330, July 16-21, 2006.

[14] T.K. Das and G.K. Venayagamoorthy, "Bio- inspired Algorithms for the design of Multiple Optimal Power system stabilizer: SPPSO and BFA," *IEEE conf. on Industry Applications,* pp. 635-641, 2006.

[15] W. J. Tan and Q. H. Wu, "A Bacterial Swarming Algorithm For Global Optimization." *Proc*. of *IEEE Congress on Evolutionary Computation*, pp. 1207-1212, 2007.

[16] L. Dos Santos Coelho, C. Da Costa Silveria, C.A. Sierakowski and P. Alotto, "Improved Bacterial Foraging Strategy Applied to TEAM Workshop Benchmark Problem." *IEEE Trans. On Magnetics, Vol. 46, no.* 8, pp. 2903-2906, Aug. 2008.

[17] A. Abraham, A. Biswas, S. D and Swagatam Das, "Analysis of Reproduction Operator in Bacterial Foraging Optimization Algorithm." *Proc. of IEEE Congress on Evolutionary Computation,* pp. 1476-1483, 2008.

[18] Ying Chu, Hua Mi, Huilian Liao, Zhen Ji, and Q. H. Wu, "A Fast Bacterial Swarming Algorithm For High-dimensional Function Optimization." *Proc. of IEEE Congress on evolutionary computation,* pp. 3135-3140, 2008.

[19] Janyl Jumadinova and Prithviraj Dasgupta, "Firefly-inspired Synchronization for Improved Dynamic Pricing in Online Markets." *Proc of Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, 2008

[20] Swagatam Das, Ajith Abraham, and Amit Konar, "Particle Swarm Optimization and Differential Evolution Algorithms Technical Analysis, Applications and Hybridization Perspectives", *Studies in Computational Intelligence, (SCI)* 116, 1-38 2008.

[21] Wang Ling, Liu Bo, "Particle Swarm Optimization and Scheduling Algorithms", Beijing: Tsinghua Publishing Company, ch.2, 2008.

[22] W.J. Tang, M.S. Li, Q.H. Wu and J.R. Saunders, "Bacterial Foraging Algorithm for Optimal Power Flow in Dynamic Environments" , *IEEE Trans. on Circuits and Systems I*, vol.55, pp. 2433-2442, Sep. 2008.

[23] S. Das, S. Dasgupta, A. Biswas and A. Abraham, "On Stability of the Chemotactic Dynamics in Bacterial-Foraging Optimization Algorithm." *IEEE Trans. on Systems, Man, and cybernetics, part: A: Systems and Humans*, vol. 39, no. 3, pp. 670-679, May 2009.

[24] LI Zhi-Jie, Liu Xiang-Dong, Duan Xiao-Dong, Wang Cuhn-Rui, "An Improved Particle Swarm Algorithm for Search Optimization." *Proc. of IEEE Global Congress on Intelligent System*, pp.154-158, 2009.

[25] Lin Cui, Hongpeng Wang, "Reach back Firefly Synchronicity with Late Sensitivity Window in Wireless Sensor Networks." *Proc. of IEEE Ninth International Conference on Hybrid Intelligent Systems*, pp. 451-456, 2009.

[26] Yang, X. S, "Firefly algorithms for multimodal optimization. Stochastic Algorithms Foundation and Applications." *SAGA 2009, Lecture Notes in Computer Sciences*, 5792, pp.169-178, 2009.

[27] Jiang, T.W, "The application of image thresholding and vector quantization using honey bee mating optimization." *Master thesis of National Ping Tung Institute of Commerce*, 2009.

[28] S. Das, S. Dasgupta, A. Biswas, A. Abraham and A. Konar, "On Stability of the Chemotactic Dynamics in Bacterial-Foraging Optimization Algorithm", *IEEE Trans. on Systems, Man and Cybernetics, Part A: Systems and Humans*, vol. 39, pp. 670-679, May. 2009.

[29] Anders Lyhne Christensen, Rehan O'Grady, and Marco Dorigo, "From Fireflies to Fault-Tolerant Swarms of Robots*." Trans. of IEEE on Evolutionary Computation*, Vol. 13, No. 4, Aug. 2009.

[30] Ming-Huwi Horng and Ting-Wei Jiang, "Multilevel Image Thresholding Selection based on the Firefly Algorithm*." Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, pp. 58-63, 2010.

[31] Juan Luis Fernandez-Martınez and Esperanza Garc´ıa-Gonzalo, "Stochastic Stability Analysis of the Linear Continuous and Discrete PSO Models", *IEEE Trans. on Evolutionary Computation*, 2010.

[32] I.A. Farhat and M.E. El-Hawary, "Dynamic adaptive bacterial foraging algorithm for optimum economic dispatch with valve-point effects and wind power", *IET Trans. on Generation, Transmission & Distribution*, vol.4, pp. 989-999, Sep. 2010.

PAPER PUBLISHED:

1. National Conference on Converging Technologies beyond 2020(CTB-2020), April 6,7 2011, Organized by: University Institute of Engineering and Technology, Kurukshetra University, Kurukshetra

2. Paper communicated to IEEE transactions on evolutionary computation.