I.J. Modern Education and Computer Science, **2014, 1, 1-16**
Published Online January 2014 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2014.01.01

Improvised Scout Bee Movements in Artificial Bee Colony

Tarun Kumar Sharma
Amity Institute of Information Technology, Amity University Rajasthan, India
E-mail: taruniitr1@gmail.com

Millie Pant
Department of Applied Science and Engineering, IIT Roorkee, India
E-mail: millidma@gmail.com

Abstract **—**In the basic Artificial Bee Colony (ABC) algorithm, if the fitness value associated with a food source is not improved for a certain number of specified trials then the corresponding bee becomes a scout to which a random value is assigned for finding the new food source. Basically, it is a mechanism of pulling out the candidate solution which may be entrapped in some local optimizer due to which its value is not improving. In the present study, we propose two new mechanisms for the movements of scout bees. In the first method, the scout bee follows a non-linear interpolated path while in the second one, scout bee follows Gaussian movement. Numerical results and statistical analysis of benchmark unconstrained, constrained and real life engineering design problems indicate that the proposed modifications enhance the performance of ABC.

Index Terms **—** ABC, Artificial Bee Colony, Quadratic Interpolation, Gaussian distribution.

I. INTRODUCTION

The past few decades several nature inspired algorithms (NIA) have emerged as a potential tool for solving global optimization problems. Global optimization is an active area of research as most of the real life problems occurring in diverse areas can be modeled as optimization problems. More than often, it is not only desirable but also necessary to obtain the global optimum rather than a local optimum. We may categorize the global optimization methods as traditional and nontraditional. Traditional methods include the classical techniques like gradient based methods (steepest descent, Newton and quasi Newton methods etc) while the nontraditional ones include Genetic Algorithms, Ta-boo Search, Simulated Annealing methods. Basically the traditional methods depend largely on the mathematical properties of the objective function and the search domain and therefore have a restricted application.

Nontraditional methods for global optimization have become more popular because of their generic nature which leads to wide applicability of these algorithms. Also it has been observed that these algorithms are also capable of locating the global optimum with a higher probability.

In the past few decades many nontraditional algorithms have been developed most of which are inspired by some natural phenomena.

The present study deals with artificial bee colony algorithm (ABC) a new computational technique proposed by Karaboga [1], based on the foraging behavior of honey bee swarm. ABC comes under the umbrella term of Swarm Intelligence (SI) algorithms which mimic the social behavior displayed by various species. Popular algorithms belonging to SI group are Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) and ABC, the recent addition to SI group.

The performance of ABC is competitive to other population-based algorithms with an advantage of employing fewer control parameters [2 - 3], simplicity and ease of implementation. ABC has captured much attention from researchers and has been applied to solve many practical optimization problems [4 - 8], since its invention in 2005. However, similar to other nontraditional optimization methods, ABC also has some drawbacks which hamper its performance. For example, the convergence speed of ABC is typically slower than some other population-based algorithms like differential evolution (DE) [9 - 10] and PSO [11] when handling unimodal problems [6]. Also, ABC can easily get trapped in the local optima when solving complex multimodal problems [6]. It has also been observed that the search equation of ABC used to generate new candidate solutions, based on the information of previous solutions is good at exploration but is poor at exploitation [12], which results in an imbalance between exploration and exploitation. Therefore, accelerating convergence speed and avoiding the local optima have become two important and appealing goals in ABC research. A number of ABC variants have, been pro-posed in literature to achieve these two goals [12 - 14].

ABC is based on the idea of division of labor in the colony of honey bees, (consisting of employed bees, on-looker bees and scout bees) for the search of potential food sources. In this paper a variation is made in the movement of scout bees. In the basic ABC algorithm, scouts are assigned a random location for determining the new food location. This is generally done with the help of computer generated random numbers following uniform distribution, which may not prove to be very efficient for locating new food sources. In the present study we focus on enhancing the movement of scout bees in order to get more efficient food locations. Two enhancements are proposed in the scout bee phase (1) quadratic interpolation and (2) Gaussian movements. The first variant is named as QABC, while the second variant is named as GABC.

Here, we would like to mention that a part of this work has been published in a conference proceeding [15] but in the present paper we have sufficiently extended it by incorporating constrained and real life problems and have done a detailed analysis of the work.

The rest of the paper is organized as follows: section 2 gives an overview of ABC algorithm. In section 3, the proposed QABC and GABC are described. In section 4, experimental settings, evaluation criteria and results are given. Finally the paper concludes with section 5.

II. SURVEY OF LITERATURE

2.1 Unconstrained ABC

ABC classifies the foraging artificial bees into three groups, namely, employed bees, onlooker bees and scout bees. Half of the colony consists of employed bees, and the other half includes onlooker bees. In the foraging process of honeybee colonies, initially, some bees search randomly for food in a given area around the hive. After finding a food source, these bees take some nectar back to the hive, deposit the nectar and share the nectar information of the food sources with other bees waiting at the *dance area* (where waggle dance, Figure 1, is performed) within the hive.

The bee colony then enters a new cycle of iterations. At each iteration, following steps take place:

(1) After sharing the information, an employed bee will either become an onlooker after abandoning the food source or continue to forage its previously visited site;

(2) Some onlookers in the hive will simultaneously follow some employed bees based on the received information in order to further forage on some specific memorized food sources; and

(3) Some scouts will spontaneously start a random search.

An important stage of the ABC algorithm, from which in fact the collective intelligence arises, is the sharing of information. This is achieved by influencing the behavior of onlookers which select their food source according to following probability P_i:

$$P_i = f_i \Big/ \sum_{k=1}^{SN} f_k \tag{1}$$

Where f_i is the fitness value of i^{th}, food source (position in parameter space). In other words onlookers will explore promising locations with higher probability than others. Candidate food sources are generated from memorized ones according to:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{i,j} - x_{k,j}) \tag{2}$$

where $i, k = 1, ..., SN, j = 1, ..., n,$ and v_i is the new food source generated by using both, the current food source x_i and a randomly chosen food source x_k from the population and $-1 \le \phi_{ij} \le 1$ (generated randomly every time it is used) determines the step size of the movement. Both, *i* and *j* are generated randomly such that $k \neq i$. When a source does not improve after a certain number of iterations, it is abandoned and replaced by the one found by a scout bee, using equation 2: which involves the generation of a new solution at random.

$$x_{i,j} = x_{\min,j} + rand(0,1)(x_{\max,j} - x_{\min,j}) \tag{3}$$

where i = 1,2,..,SN. j = 1, 2,…, n. $x_{\max}$,j and $x_{\min,j}$ are upper and lower bounds of parameter j, respectively. These food sources are randomly assigned to SN number of employed bees and their finesses are evaluated.

Figure.1: Waggle dance of Honey bee in the hive

Basic steps of Artificial Bee Colony:

Initialization of food sources (Population): The initial population of solutions is filled with *SN* number of randomly generated *n*-dimensional real-valued vectors (i.e., food sources). Let $X_i = \{x_{i,1}, x_{i,2}, ..., x_{i,n}\}$ represent the i^{th} food source in the population, and then each food source is generated by equation (3).

Employed bee initialization: In this phase each employed bee X_i generates a new food source V_i in the neighborhood of its present position by using solution search equation (2). Once V_i is obtained, it will be evaluated and compared to X_i. If the fitness of V_i is equal to or better than that of X_i, V_i will replace X_i and become a new member of the population; otherwise X_i is retained. In other words, a greedy selection mechanism is employed between the old and candidate solutions.

Probabilistic Selection: An important stage of the ABC algorithm, from which in fact the collective intelligence arises, is the sharing of information. This is achieved by influencing the behavior of onlookers which will select their food source according to probability equation (1)

Onlooker bee phase: An onlooker bee evaluates the nectar information taken from all the employed bees and selects a food source X_i depending on its probability value P_i. Once the onlooker has selected her food source X_i, she produces a modification on X_i by using equation (2). As in the case of the employed bees, if the modified food source has a better or equal nectar amount than X_i, the modified food source will replace X_i and become a new member in the population.

Scout bee phase: If a food source X_i cannot be further improved through a predetermined number of trials limit, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout produces a food source randomly using equation (3).

2.2 Constrained ABC

ABC algorithm for solving constrained optimization problems, we adopted the Pareto ranking method instead of the selection process (greedy selection) of the ABC algorithm described in the previous section. The following criteria are always enforced:

- Any feasible solution is preferred to any infeasible solution,
- Among two feasible solutions, the one having better objective function value is preferred,
- Among two infeasible solutions, the one having smaller constraint violation is preferred.

Because initialization with feasible solutions is very time consuming process and in some cases it is impossible to produce a feasible solution randomly, the ABC algorithm does not consider the initial population to be feasible. Structure of the algorithm already directs the solutions to feasible region in running process due to the selection process. Scout production process of the algorithm provides a diversity mechanism that allows new and probably infeasible individuals to be in the population. In order to produce a candidate food position from the old one in memory, the adapted ABC algorithm uses the following expression:

$$v_{ij} = \begin{cases} x_{ij} + \phi_{ij}(x_{ij} - x_{kj}), & if \;\; R_j \le MR \\ x_{ij} \;, & otherwise \end{cases} \tag{4}$$

where $k \in \{1, 2, ..., SN\}$ is randomly chosen index. Although k is determined randomly, it has to be different from i. R_j is randomly chosen real number in the range [0,1] and $j \in \{1, 2, ..., D\}$. MR, modification rate, is a control parameter that controls whether the parameter x_{ij} will be modified or not. In the version of the ABC algorithm proposed for constrained optimization problems, artificial scouts are produced at a predetermined period of cycles for discovering new food sources randomly. This period is another control parameter called scout production period (*SPP*) of the algorithm. At each *SPP* cycle, it is controlled if there is an abandoned food source or not. If there is, a scout production process is carried out.

Detailed algorithm is discussed in Figure 2.

```
Initialize $p_w$, the percentage of employed bees
Initialize $n_e^{max}$ maximum number of explorer bees
Initialize $\lambda$, the foraging limit
Set $n_s$ as the size of swarm
Set the fittest bee, $\beta$ to null
Create and initialize to random positions $n_w = n_s p_w$ employed bees
Set the number of onlooker bees, $n_o = n_s - n_w$
for each worker bee, $w_i$ do
        Set $a_i = 0$, where $a_i$ is the number of failed position updates
end
while stopping condition is false do
        Set $n_e = 0$, where $n_e$ is the number of explorer bees
        for $i = 1,\ldots,n_w$ do
                Randomly select $j \in [1, n_w], j \ne i$
                Create a new position $v_{wiwj}$ from worker bees using equation (4)
                if $f(v_{wiwj}) < f(x_{wi})$ based on Pareto Ranking Method then
                        $x_{wi} = v_{wiwj}$
                end
                else
                        $a_i$++
                end
                if $a_i > \lambda$ and $n_e < n_e^{max}$ then
                        $n_e$++
                        Move $w_i$ to a new random position in the search space
                        $a_i = 0$
                end
        end
        for $i = 1,\ldots, n_o$ do
                Select $j \in [1, n_w]$ proportionate to $f(x_{wi})$
                Set $o_i = w_j$
                Select $k \in [1,\ldots,n_w], k \ne j$
                Create a new position $v_{oiwj}$ from onlooker bee $o_i$ and worker bee $w_j$ using equation (4)
                if $f(v_{oiwj}) < f(x_{oi})$ based on Pareto Ranking Method then
                        $x_{oi} = v_{oiwj}$
                end
        end
        for each bee, $b_i$, in the swarm do
                if $f(x_{bi} < f(x_\beta)$ then
                        $\beta = b_i$
                end
        end
end
Return $x_\beta$ as Solution.
```

Figure.2: Algorithm: Constrained Artificial Bee Colony

III. QABC AND GABC: THE PROPOSED VARIANTS

The proposed QABC and GABC differ from the basic ABC in terms of the movements of the scout bees.

Instead of assigning a random movement for the scout bees, in the proposed variants the scout bees are moved in order to find a better location for them. For this purpose we have used the method of interpolation and method following Gaussian distribution.

Quadratic Interpolation: In the present study, the three initial solution vectors x_{r1}, x_{r2}, and x_{r3} are selected randomly between 0 and 1, distinct from each other. From these three points, the coordinates of the new Food Location $V_{i,G+1} = (v_{1,G+1}, v_{2,G+1}, ..., v_{n,G+1})$, are determined as:

$$v_{i,g+1} = 0.5 * \frac{(x_{r_2,g}^2 - x_{r_3,g}^2) * f(x_{r_1,g}) + (x_{r_3,g}^2 - x_{r_1,g}^2) * f(x_{r_2,g}) + (x_{r_1,g}^2 - x_{r_2,g}^2) * f(x_{r_3,g})}{(x_{r_2,g} - x_{r_3,g}) * f(x_{r_1,g}) + (x_{r_3,g} - x_{r_1,g}) * f(x_{r_2,g}) + (x_{r_1,g} - x_{r_2,g}) * f(x_{r_3,g})} \quad (4)$$

Gaussian movement: Gaussian distribution also called a “bell shaped curve” as a mode of perturbation (or mutation) has been used for generating new candidate solutions. The PDF for Gaussian function is given as: $f(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$; with mean 0 and standard deviation 1, i.e. N (0, 1).

Proposed Variants: After initialization QABC starts like the usual ABC as discussed in previous section and when the food source X_i cannot be further improved through a fixed trial limit, the food source get abandoned, and the corresponding employed bee act as *scout bee*. Than scout bee produces a new food source using, the quadratic interpolation given in equation (5) for QABC. In GABC the scout bee produces a new food source using, Gaussian random numbers and the equation is given as:

$$x_{i,j} = x_{\min,j} + Gauss(-1,1)(x_{\max,j} - x_{\min,j}) \quad (5)$$

where, *Gauss* is a random number following Gaussian distribution. Fig. 3 explains the pseudocode of QABC and GABC & flow graph is shown in Fig. 4.

```
Begin
  1.  Initialize the population of food sources $x_i$, $i$ = 1,…,SN
  2.  Evaluate each food source $x_i$, $i$ = 1,…,SN
  3.  cycle = 1
Repeat
For each food source $x_i$ in the population
  4.  Generate a new food source $v_i$ by its corresponding
      employed bee (equation (2))
  5.  Evaluate $v_i$
  6.  Keep the best solution between $x_i$ and $v_i$
End
  7.  Select, based on fitness proportional selection, the food
      sources to be visited by onlooker bees
For each food source $x_i$ chosen by an onlooker bee
  8.  Generate a new food source $v_i$ by its corresponding
      onlooker bee (equation (2))
  9.  Evaluate $v_i$
  10. Keep the best solution between $x_i$ and $v_i$
End
  11. Use the scout bee to replace those abandoned food
      sources using (equation (4) for QABC and equation (5)
      for GABC)
  12. Keep the best solution between $x_i$ and $v_i$
  13. Save in memory the best food source so far
  14. cycle = cycle + 1
Until cycle
End
```

Figure.3: Pseudocode of Proposed QABC & GABC

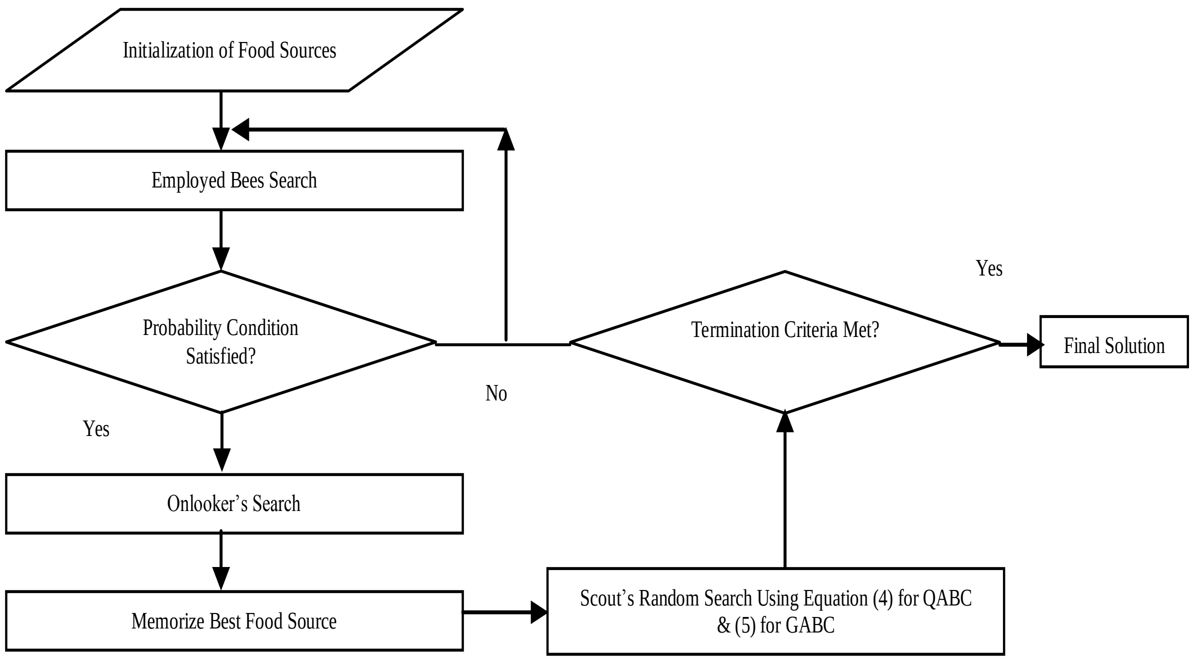

Initialization of Food Sources
Employed Bees Search
Probability Condition Satisfied?
Yes
No
Onlooker’s Search
Memorize Best Food Source
Scout’s Random Search Using Equation (4) for QABC & (5) for GABC
Termination Criteria Met?
Yes
Final Solution

Figure.4: Flow Graph of Proposed Variants

IV. EXPERIMENTAL SETTINGS, PERFORMANCE CRITERIA AND RESULTS

Benchmark and Engineering Problems: In order to assess the performance of the proposed QABC and GABC algorithm, we considered four unconstrained, six constrained and five engineering design given in Appendix I, II and III respectively.

Structure of the Problems: The general NLP is given by nonlinear objective function f, which is to be minimized/maximized with respect to the design variables $\bar{x} = (x_1, x_2, ..., x_n)$ and the nonlinear inequality and equality constraints. The mathematical models of the problems considered in the paper are of the type:

$$\begin{aligned} &Minimize / Maximize\ f(\bar{x}) \\ &Subject\ to: \quad g_j(\bar{x}) \le 0, \quad j = 1, ..., p \\ &\qquad h_k(\bar{x}) = 0, \quad k = 1, ..., q \\ &\qquad x_{i\min} \le x_i \le x_{i\max} \ (i = 1, ..., n) \end{aligned}$$

where p and q are the number of inequality and equality constraints respectively.

A set of six constrained benchmark problems taken from literature [16] is considered to evaluate the performance of the proposed QABC and GABC. All the problems are nonlinear in nature i.e. either the objective function or the constraints or both have a nonlinear term in it. The mathematical models of the problems along with the optimal solution are given in Appendix II.

Characteristics of test functions (*f*)

f	n	Func.Type	ρ (%)	li	ni	le	ne	a
g01	13	Quad.	0.0003	9	0	0	0	6
g02	20	Nonl.	99.9973	2	0	0	0	1
g03	10	Nonl.	0.0026	0	0	0	1	1
g04	5	Quad.	27.0079	4	2	0	0	2
g05	4	Nonl.	0.0000	2	0	0	3	3
g06	2	Nonl.	0.0057	0	2	0	0	2

Parameter *n* denotes the number of parameters. The function can be linear, nonlinear (Nonl.) or quadratic (Quad.), *li* is the number of linear inequality constraints, *ni* is the number of nonlinear inequality constraints, *le* is the number of linear equality constraints, *ne* is the number of nonlinear inequality constraints, *a* is the number of active restrictions and *ρ* is a percentage of the feasible area. A percentage of feasible area is:

$$\rho = |F|/|S| \tag{6}$$

where |*F*| is the number of feasible solutions and |*S*| is the total number of solutions randomly generated. Michalewicz and Schoenauer [17] suggested a total number of 1,000,000 solutions for |*S*|.

Engineering Design Problems: The credibility of an optimization algorithm also depends on its ability to solve real life/engineering's problems. In this paper we took five real life engineering design problems to validate the efficiency of all the proposed algorithms.

Mathematical models of problems are given in Appendix III.

Experimental Settings: The proposed algorithms are tested on 4 unconstrained, 6 constrained benchmark problems and 5 engineering problems given in Appendix I, II and III respectively. The following control parameters are: colony size (SN), MCN (Maximum Cycle Numbers) and "*limit*". ABC and the proposed variants are implemented on Dev-C++ and the experiments are conducted on a computer with 2.00 GHz Intel (R) core (TM) 2 duo CPU and 2- GB of RAM. For each problem, all the algorithms independently run 30 times. The parameter setting is taken as follows:

Parameter	**Values**
Unconstrained Problems	
Population size	100
limit	100
Value to Reach (VTR)	10^{-15}
Maximum MCN	8000
Constrained and Engineering Design Problems	
Colony Size	20
Limit	MCN/(2*SN)
Max. Cycle Numbers (MCN)	6000
NFE (for Engg. desing. Prob)	5.0E+06
Max. Func. Evaluations (Max_FEs)	6.0E+05
MR (Modification Rate) (Akay and Karaboga, 2010)	0.4
Runs	30

Performance Criteria: *Mean Fitness, Standard Deviation, Best and Worst:* The average of function fitness value that an algorithm can find, using predefined MCN, is recorded in each run and then average of the function fitness values are calculated. Also the average, standard deviation, best and worst of the fitness values are calculated.

MCN: The MCN (Maximum Cycle Number) is recorded when the VTR is reached before to reach maximum MCN. i.e. we set the termination criteria as $|f_{optimal} - f_{global}| \le VTR$ and record MCN over 30 runs.

Acceleration rate (AR) in %: This criterion is used to compare the convergence speeds between ABC, GABC and QABC. It is defined as follows:

$$AR = \frac{MCN_{ABC,GABC} - MCN_{QABC}}{MCN_{ABC,GABC}} \%$$

Comparison Criteria, Results and Discussion: *Comparison Criteria:* We used several criteria to measure the performance of the proposed QABC and GABC algorithm and to compare it with basic ABC. In Tables 1 – 2 we recorded the performance the proposed QABC and GABC for unconstrained problems in terms of best worst and average fitness function value along with the standard deviation (Std) while increasing the MCN to two different values 5000 with D=30 & SN=20 and 8000 with D=50 & SN=50. t-Test and acceleration rate is given in Table 3.

In Tables 4 and 5 the performance of QABC and GABC is compared with basic ABC for solving constrained optimization problems. The comparison criteria for all the algorithms taken in the present study are given as:

1. Feasible Run: A run during which at least one feasible solution is found in Max NFE.
2. Successful Run: A run during which the algorithm finds a feasible solution ~x satisfying (f(x) – f(x*) <= 0.0001 ·
3. Feasible Rate = (# of feasible runs) / total runs
4. Success Rate = (# of successful runs) / total runs
5. Success Performance = mean (FE's for successful runs) * (# of total runs) / (# of successful runs)

Further, the average acceleration rate (AR_{avg}) and the average success rate (SR_{avg}) over test functions are calculated as follows:

$$AR_{avg} = \frac{1}{n}\sum_{i=1}^{n} AR_i$$

$$SR_{avg} = \frac{1}{n}\sum_{i=1}^{n} SR_i$$

In order to get a better insight into the relative performance of ABC, QABC, GABC, [18] and [19], the value of their performance is calculated. This performance gives specified importance to the number of function evaluations to observe the efficiency of the algorithm. For the computational algorithms under comparison the value of performance P_j for the j^{th} algorithm is computed as under:

$$P_j = \frac{1}{N}\Sigma_{i=1}^{N}(\alpha_1^i)$$

$$where\ \alpha_1^i = \begin{cases} \dfrac{Mf^i}{Af^i}, & if\ \ Sr^i \geq 0, \\ 0, & if\ \ Sr^i = 0 \end{cases}$$

$$where\ i = 1,2...,N.$$

Here, Af^i is the average number of function evaluations used by the j^{th} algorithm in obtaining the optimal solution of i^{th} problem in the case of the successful runs, and Mf^i the minimum of the average number of function evaluations of successful runs used of the algorithms under comparison in obtaining the optimal solution of i^{th} problem. N is total number of problems on which the performance of algorithms has been tested.

The larger the value of $P_{j,}$ better is the performance of the algorithm.

For engineering design problems the performance of ABC and proposed variants is compared in terms of mean fitness value & standard deviation (Std.) given in Table 6 and in Table 7 the iterations and acceleration rate is given.

Results and Discussions: *Unconstrained Problems*: In Table 2 we have taken the results on the basis of average error. In this case MCN is fixed at 5000 and 8000, to estimate the average of minimum fitness function value in 30 runs. From the Table 2 it can be clearly observed that for all benchmark functions QABC gives better results than GABC and ABC. A two tail sample *t-test* is also applied to analyze the statistical significance of the proposed algorithm. We have checked the significant difference of QABC with respect to GABC and ABC at 5% level of significance. The calculated *t-value* (Table 3) of all function is greater than *t-table* value that shows the significantly better performance in the comparison of ABC. In the Table-2, we fixed *VTR* as given in experimental setting and then calculated the MCN of 30 runs. From Table-2 we can see that the proposed QABC gives the better results for every function in the comparison to the other algorithms. From the Table-3 it is clear that the proposed QABC is faster than GABC by 13.50% and ABC by 16.50%, when MCN=5000, D=30 & SN=20 and in other case when MCN=8000, D=50 & SN=50, QABC is again faster than GABC by12.27% and ABC by 10.57%. Best and worst function values in 30 runs are also presented in Table-1.

Fig. 4(a) & 4(b) shows the convergence graph Rosenbrock's and Griekwank function. MCN taken to estimate the average of minimum fitness function value in 30 runs are also presented graphically in Fig. 5.

Constrained Problems: The results obtained in Table 4 using QABC are better or equal to that of results obtained by original ABC, GABC algorithm and MO-ABC for constrained optimization problems. The *g*05 function illustrates that the QABC due to greater exploration capability, better best results are reached, but also the worst result is slightly worse than compared algorithm. The standard deviation for *g02* function is also inferior for the same reason. QABC reaches much better results for *g*02 and *g*03 function then the original ABC algorithm.

The superior performance of QABC is more visible from Tables 4 and 5 where the results are recorded after fixing the accuracy at 0.0001. In these tables we can see that the proposed QABC gave a better or at par performance with the other three algorithms. We will now take the comparison criteria one-by-one and discuss them briefly. The first criterion is that of a feasible run. A run is said to be feasible if at least one feasible solution is obtained in maximum number of function evaluations. According to this criterion all the algorithm gave 100% feasible rate for all the test problems.

However, if we observe the second criterion which is of successful run and is recorded when the algorithm finds a feasible solution satisfying the given accuracy (=0.0001) it can be seen that the proposed QABC outperforms the other algorithms in all the test cases. The third criterion is that of the success performance which depends on the feasibility rate and success rate, as described in the previous subsection. Here also QABC gave a better performance in comparison to the other three algorithms taken for comparison.

Comparative acceleration rate is also shown graphically in Fig. 6 using radar representation, which explains that QABC is 4.18% better than ABC, 3.64% better than GABC, 58.11% better than [18] where as 68.7% better than [19] for *g01*. Similarly the graphs explain the acceleration rate for rest of the problems *g02* - *g06*.

The performance comparison graph of algorithms is given in Fig. 7 (a). This curves (best solution versus NFE's) show that QABC converges faster than others toward the optimal solution. Also Average Success performance and acceleration performance graph to solve six constrained benchmark problems are given in single graph shown in Fig. 7(b).

Engineering Design Problems: The results of engineering design problems are presented in Table 6 and 7. We see that in terms of average fitness function value and standard deviation all the algorithms gave more or less similar results although in some cases the proposed algorithms gave a marginally better performance than basic ABC and GABC. From Table 7 it can be clearly observed that QABC emerges clear winner in terms of iterations taken to reach the optimum value, further QABC is 24.36% faster than ABC where as 8.24% faster than GABC when compared in terms of acceleration rate.

TABLE 1: Mean, Standard Deviation, Best, Worst, Mean and values obtained by ABC, GABC and QABC through 30 independent runs on function from f_1 to f_4

F	Algorithm	MCN=5000/D=30/SN=20		MCN=8000/D=50/SN=50	
		Mean Std. Dev	**Best Worst**	**Mean Std. Dev**	**Best Worst**
f_1	ABC	1.162e-015 1.896e-015	0.062e-015 2.235e-014	2.841e-014 3.867e-014	1.507e-014 4.654e-014
	GABC	8.295e-016 2.531e-017	1.460e-016 3.310e-014	3.159e-015 9.268e-016	1.679e-016 5.130e-017
	QABC	1.437e-016 4.448e-017	8.377e-016 4.626e-014	2.551e-014 0	1.962e-014 4.818e-014
f_2	ABC	3.750e-015 6.297e-014	9.277e-016 5.245e-012	1.379e-012 4.167e-013	1.001e-013 1.712e-014
	GABC	1.053e-015 7.870e-015	2.894e-016 5.277e-013	1.313e-014 4.578e-015	1.065e-015 1.579e-016
	QABC	4.256e-016 1.406e-015	2.976e-016 6.041e-015	1.036e-014 3.838e-015	1.075e-015 1.633e-016
f_3	ABC	2.016e-002 8.446e-003	1.953e-002 3.337e-002	1.075e-001 0	1.056e-002 3.458e-003
	GABC	2.738e-004 8.152e-004	1.199e-004 7.972e-004	1.181e-002 0	0.326e-002 2.707e-003
	QABC	2.264e-004 8.389e-005	9.163e-005 5.065e-001	9.753e-003 0	2.808e-004 2.983e-003
f_4	ABC	1.514e-016 4.309e-017	8.289e-017 1.514e-014	1.147e-015 3.265e-016	0.472e-016 1.147e-017
	GABC	3.176e-017 9.039e-017	1.982e-018 3.691e-016	7.155e-017 2.036e-018	3.673e-018 7.155e-019
	QABC	8.206e-018 1.333e-019	2.549e-020 1.355e-017	1.670e-017 2.183e-018	6.006e-018 7.670e-019

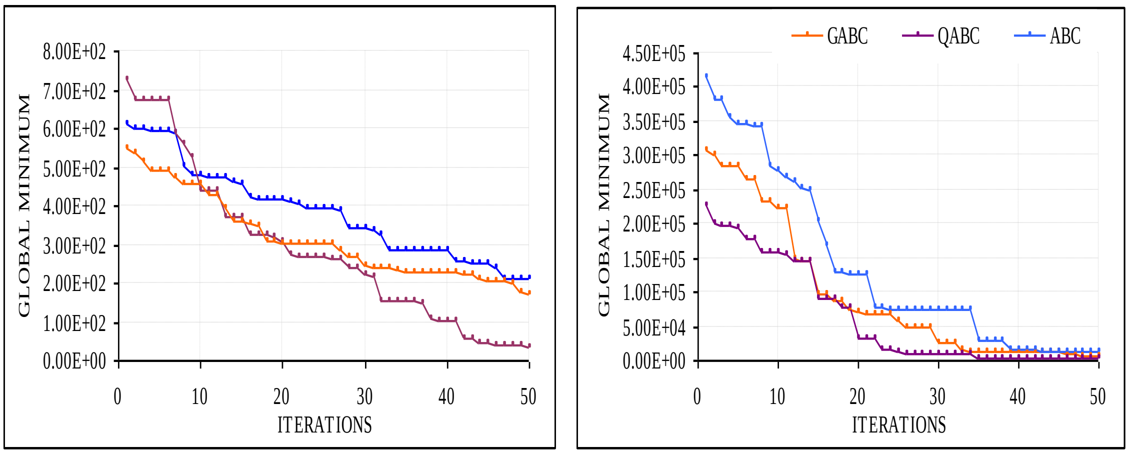
TABLE 2: MCN taken by functions for VTR (NC-Not Converge)

F	D=30/SN=20			D=50/SN=50		
	ABC	GABC	QABC	ABC	GABC	QABC
f_1	1148	1102	1069	1737	1770	1255
f_2	4011	3903	1928	7023	6552	5790
f_3	NC	NC	NC	NC	NC	NC
f_4	1083	1009	1005	1674	1636	1610

TABLE 3: T-test and AR(%), (here 3/1 implies QABC /ABC, and 3/2 implies QABC /GABC)

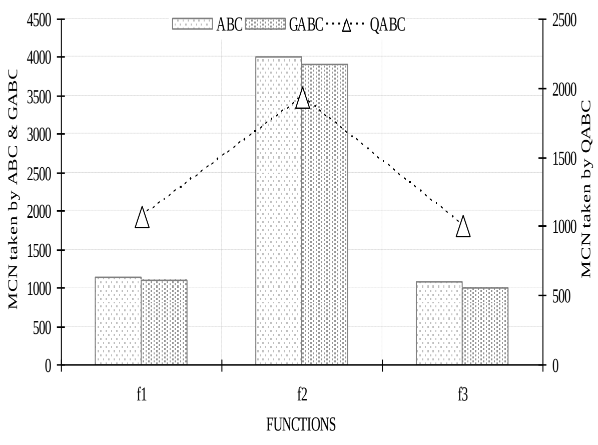
F	t-test				AR			
	SN=20/D=30		SN=50/D=50		SN=20/D=30		SN=50/D=50	
	3/1	3/2	3/1	3/2	3/1	3/2	3/1	3/2
f_1	1.611	40.20	0.224	72.35	6.9	3.0	27.7	29.1
f_2	0.16	0.235	9.853	1.391	51.9	50.6	17.6	11.6
f_3	7.080	0.173	1.#NF	10.385	NC	NC	NC	NC
f_4	9.969	0.781	1.#NF	55.123	7.2	0.4	3.8	1.6

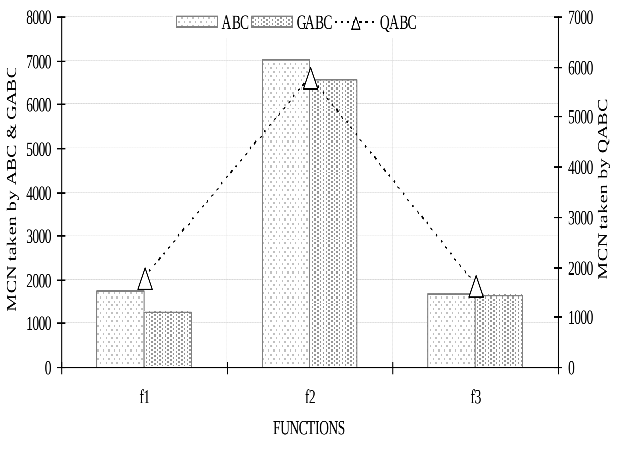
1.#NF indicates std. dev. = 0; NC Not Converge



(a) (b)

Figure.5: Convergence plot of (a) Griekwank (b) Rosenbrock





(a) (b)

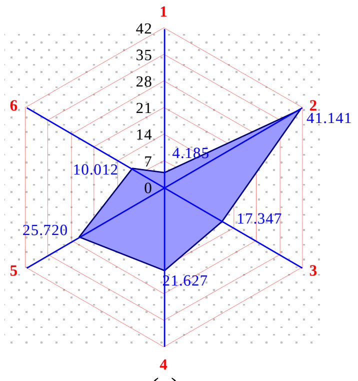
Figure.6: MCN taken by function f_1, f_2 & f_3 (a) D=30, SN=20 (b) D=50, SN=50

TABLE 4: Simulation results for the constrained optimization problems

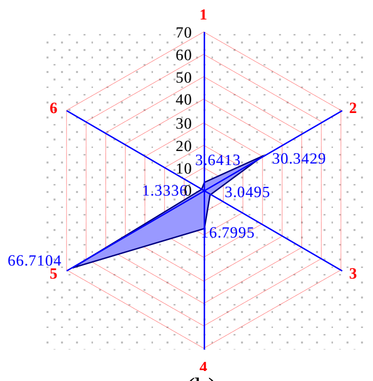
f	**Statistics**	**Algorithm**			
		ABC	**MO-ABC**	**QABC**	**GABC**
g01	Mean	-15	-15	-15	-15
	SD	0	0	0	1.01E-14
	Worst	-15	-15	-15	-15
	Best	-15	-15	-15	-15
g02	Mean	0.792412	0.793506	0.799731	0.80357
	SD	0.012	0.014	0.027	0.00481
	Worst	0.749797	0.744311	0.799431	0.794531
	Best	0.803598	0.803605	0.803609	0.803614
g03	Mean	-1	-1	-1.00037	-0.7563
	SD	0	0	0	0.0217
	Worst	-1	-1	-1.00037	-0.7981
	Best	-1	-1	-1.00037	-0.8753
g04	Mean	-30665.539	-30665.539	-30665.531	-30665.538
	SD	0	0	0	0
	Worst	-30665.539	-30665.539	-30665.539	-30665.539
	Best	-30665.539	-30665.539	-30665.539	-30665.539
g05	Mean	5185.714	5162.496	5125.9462	5126.5067
	SD	75.358	47.8	52.21	51.40
	Worst	5438.387	5229.134	5232.179	5126.4967
	Best	5126.484	5126.582	5126.531	5126.4967
g06	Mean	-6961.814	-6961.814	-6961.805	-6961.8138
	SD	0.002	0	0.00001	1.98E-15
	Worst	-6961.805	-6961.814	-6961.8129	-6961.8138
	Best	-6961.814	-6961.814	-6961.8139	-6961.8138

TABLE 5: Comparison Results: NFE (Best, Worst and Mean) to achieve the fixed accuracy level ((f(x) – f(x*)) <= 0.0001), success rate, Feasible Rate and Success Performance for problems g01 – g06. Algo: Algorithms

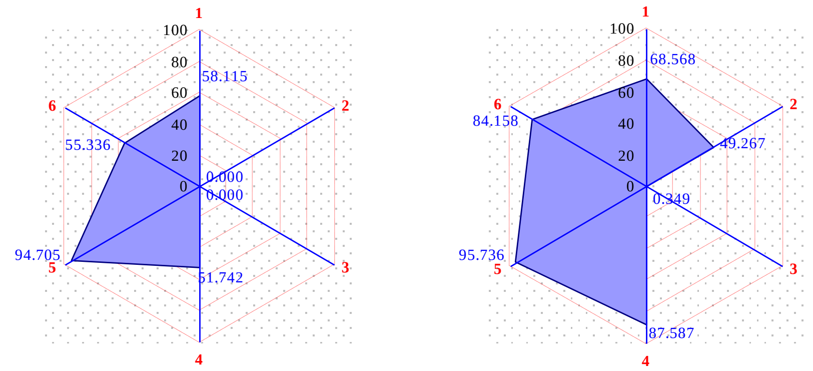
Problem	**Algo**	**Best**	**Worst**	**Mean**	**Feasible Rate (%)**	**Success Rate (%)**	**Success Performance**
*g*01	ABC	25850	86500	33308	100	100	33308
	QABC	25619	67997	31914	100	100	31914
	GABC	27981	83536	33120	100	100	33120
	[18]	25273	346801	76195	100	52	146528.8
	[19]	95100	106900	101532	100	100	101532
*g*02	ABC	175832	428719	199273	100	100	199273
	QABC	92764	192850	117290	100	100	117290
	GABC	127823	215782	168382	100	100	168382
	[18]	-	-	-	100	0	-
	[19]	180000	327900	231193	100	56	412844.6
*g*03	ABC	498266	587179	543319	100	58	936756.9
	QABC	440982	479265	449071	100	76	590882.9
	GABC	475387	511732	463196	100	59	785078
	[18]	-	-	-	100	0	-
	[19]	450100	454000	450644	100	100	450644
*g*04	ABC	11956	13651	12651	100	100	12651
	QABC	9659	10842	9915	100	100	9915
	GABC	10750	12281	11917	100	100	11917
	[18]	15363	25776	20546	100	100	20546
	[19]	74300	85000	79876	100	100	79876
*g*05	ABC	21765	91985	25964	100	76	34163.16
	QABC	13350	65400	19286	100	88	21915.91
	GABC	19573	78194	57934	100	78	74274.36
	[18]	94156	482411	364218	100	16	2276363
	[19]	450100	457200	452256	100	100	452256
*g*06	ABC	9167	10284	9948	100	100	9948
	QABC	7373	9892	8952	100	100	8952
	GABC	8791	9972	9073	100	100	9073
	[18]	16794	22274	20043	100	100	20043
	[19]	47800	61100	56508	100	100	56508



(a)

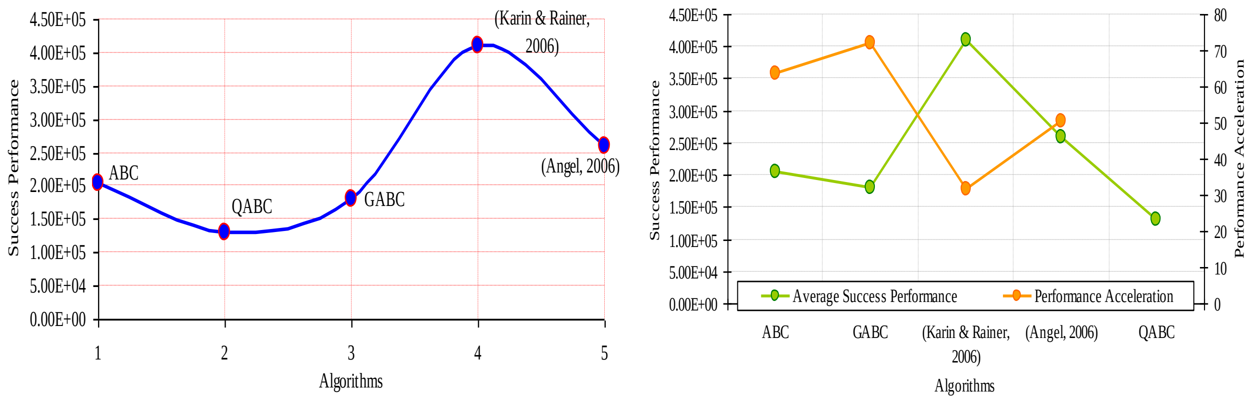


(b)



(c) (d)

Figure.7: Radar representation of acceleration rate of constrained benchmark problems (a) ABC Vs. QABC (b) GABC Vs. QABC (c) [18] Vs. QABC (d) [19] Vs. QABC



(a) (b)

Figure.8: (a) Average Success performance graph for all algorithms (b) Average Success performance and acceleration performance graph to solve six constrained benchmark problems.

TABLE 6: Mean of fitness function value and (standard deviation) and for all algorithms

F	D	ABC	GABC	QABC
		Mean (Std.)	**Mean (Std.)**	**Mean (Std.)**
F_1	2	169.849 (0.00801)	169.846 (1.7034e-016)	169.842 (1.6309.e-016)
F_2	3	4.21319 (3.1904e-016)	4.20916 (1.4291e-017)	4.20079 (.000101)
F_3	4	4.7457e-008 (1.8741e-018)	1.8536e-009 8.0873e-023)	2.9014e-009 (2.9443e-024)
F_4	10	-26.0119 (0.71933)	-26.0317 (0.03981)	-26.0418 (0.65189)
F_5	6	2.9975e+006 (0.22175)	2.9975e+006 (0.0)	2.7917e+006 (0.00319)

TABLE 7: Comparison of QABC with other algorithms in terms of Iterations and AR (%), Here 3/1 implies QABC vs. ABC and 3/2 implies QABC vs.GABC

Function	**ABC**	**GABC**	**QABC**	**AR 3/1**	**AR 3/2**
F_1	289	249	197	31.83	20.88
F_2	756	524	513	32.14	2.10
F_3	418	317	321	23.21	---
F_4	240000	240000	240000	0.00	0.00
F_5	8517	6913	5568	34.62	19.46
Average Acceleration Rate (AR(%))				24.36	8.24

V. CONCLUSION

ABC is one of the latest additions to the class SI based algorithms. In the present study enhancements are suggested in the scout bee phase of the ABC algorithm. Instead of providing a random movement for scout bees, like the basic ABC algorithm, interpolation and Gaussian movements are embedded to generate more efficient food locations. The corresponding variants named as QABC and GABC are able to explore and exploit the search space more effectively. These variants are tested on 4 unconstrained, 6 constrained benchmark problems and five engineering design problems. Various performance criteria were considered to assess the proposed algorithms. Numerical results and graphical illustrations indicate that the proposed enhancements improve the performance of basic ABC in terms of convergence rate and fitness function.

REFERENCES

[1] Karaboga D, (2005). An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Kayseri, Turkey: Erciyes University.

[2] Karaboga D, Basturk B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of Global Optimization; 39:171–459.

[3] Karaboga D, Basturk B. (2008). On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing; 8:687–97.

[4] Singh A. (2009) an artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. Applied Soft Computing; 9:625–31.

[5] Kang F, et al. (2009). Structural inverse analysis by hybrid simplex artificial bee colony algorithms. Computers & Structures; 87:861–70.

[6] Sam rat L, et al.(2010). Artificial bee colony algorithm for small signal model parameter extraction of MESFET. Engineering Applications of Artificial Intelligence; 11:1573–2916.

[7] Sharma TK, Pant M and Singh M. (2013). Nature Inspired Metaheuristic Techniques as Powerful Optimizers in Paper Industry, Materials and Manufacturing Processes, Taylor and Francis, 28(7), pp. 788 - 802.

[8] Sharma TK and Pant Mille (2013). Enhancing the Food Locations in an Artificial Bee Colony Algorithm, Soft Computing, Springer, 17(10), pp. 1939 - 1965.

[9] Storn R, Price K. (2010). Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization; 23:689–94.

[10] Storn R. and Price K. V. (1997). Differential Evolution —a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optimization, vol. 11, pp. 341–359, (1997).

[11] Eberhart, R., Kennedy, J. (1995). Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, November, pp.1114–1121.

[12] Zhu GP, Kwong S. (2010). Gbest-guided artificial bee colony algorithm for numerical function optimization. Applied Mathematics and Computation, doi:10.1016/j.amc.2010.08.049.

[13] Akay, D. Karaboga, D. (2010). A modified Artificial Bee Colony algorithm for real-parameter optimization, Information Science, doi:10.1016/j.ins.2010.07.015.

[14] Alatas B. (2010). Chaotic bee colony algorithms for global numerical optimization. Expert Systems with Applications, 37:5682–7.

[15] Sharma, T.K. and Pant, M. (2011). Enhancing Scout Bee Movements in Artificial Bee Colony Algorithm. In Proc. of SocProS - 2011, December 20-22, 2011 Advances in Intelligent and Soft Computing, Springer Berlin Heidelberg Volume 130, 2012, pp 601-610.Roorkee, India.

[16] Subotic, Milos. (2011). Artificial bee colony algorithm with multiple onlookers for constrained optimization problems. In Proceedings of the European Computing Conference, pp: 251-256.

[17] Michalewicz, Z., Schopenhauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation. Vol. 4, No. 1, pp. 1–32.

[18] Karin Zielinski and Rainer Laur. (2006). Constrained Single-Objective Optimization Using Particle Swarm Optimization. IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, pp. 443 – 450.

[19] Angel E. Mu˜noz-Zavala, Arturo Hern ́andez-Aguirre, Enrique R. Villa-Diharce and Salvador Botello-Rionda. (2006). "PESO+ for Constrained Optimization" IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, pp. 231 – 238.

[20] Beightler C., Phillips D. (1976). Applied geometric programming. John Wiley and Sons, New York.

[21] Prasad B., Saini J. (1991). Optimal thermo hydraulic performance of artificially roughened solar air heater. Journal Solar Energy, 47: 91– 96.

[22] Babu B. (2004).New optimization techniques in engineering. Springer-Verlag, Berlin Heidelberg.

[23] Chakraborthy U., Das S., Konar A. (2006). Differential evolution with local neighbourhood. IEEE Congress on Evolutionary Computation, Canada, pp. 2042 – 2049.

Appendix A

Name of function	Definition	Range	Known global optimum
Griekwank	$f_1(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$	[-600, 600]	f_1 (0,…,0) = 0.
Restrigin's	$f_2 = 10n + \sum_{i=1}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	[-5.12, 5.12]	f_2 (0…, 0) = 0.
Rosenbrock's	$f_3(x) = \sum_{i=1}^{n-1}\left[100\left(x_{i+1}^2 - x_i^2\right) + \left(1 - x_i^2\right)\right]$	[-30, 30]	f_3 (1,…,1) = 0.
Sphere	$f_4(x) = \sum_{i=1}^{n} x_i^2$	[-5.12, 5.12]	f_4 (0,…,0) = 0.

Appendix B

1. **g01:**

Minimize $f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$

Subject to:

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \le 0$$
$$g_2(x) = 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \le 0$$
$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \le 0$$
$$g_4(x) = -8x_1 + x_{10} \le 0$$
$$g_5(x) = -8x_2 + x_{11} \le 0$$
$$g_6(x) = -8x_3 + x_{12} \le 0$$
$$g_7(x) = -2x_4 - x_5 + x_{10} \le 0$$
$$g_8(x) = -2x_6 - x_7 + x_{11} \le 0$$
$$g_9(x) = -2x_8 - x_9 + x_{12} \le 0$$
$$0 \le x_i \le 1 \ (i = 1,2,...,9),\ 0 \le x_i \le 100 \ (i = 10,11,12),\ 0 \le x_{13} \le 1$$

The optimum value is $f(x^*) = -15$ at $x^* = (1,1,1,1,1,1,1,1,1,3,3,3,1)$

Constraints g_1, g_2, g_3, g_7, g_8, g_9 are active.

2. **g02**

Maximize $f(x) = \left|\frac{\sum_{i=1}^{n}\cos^4(x_i) - 2\prod_{i=1}^{n}\cos^2(x_i)}{\sum_{i=1}^{n} i x_i^2}\right|$

Subject to:

$$g_1(x) = 0.75 - \prod_{i=1}^{n} x_i \le 0$$
$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \le 0$$
$$0 \le x_i \le 10 \ (i = 1,2,...,n),\ n = 20$$

The optimum value is unknown. The known best value is $f(x^*) = 0.803619$

Constraint g_1 is active.

3. **g03**

Minimize $f(x) = -(\sqrt{n})^n \prod_{i=1}^{n} x_i$

Subject to:

$$h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$
$$0 \le x_i \le 10 \ (i = 1,2,...,n)$$

The optimum value is $f(x^*) = -1$ at $x^* = (1/\sqrt{n})$, $n = 10$.

4. **g04**

Minimize $f(x) = 5.3578547 x_3^2 + 0.8356891 x_1 x_5 + 37.293239 x_1 - 40792.141$

Subject to:

$$g_1(x) = 85.334407 + 0.0056858 x_2 x_5 + 0.0006262 x_1 x_4 - 0.0022053 x_3 x_5$$
$$g_2(x) = 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_3^2$$
$$g_3(x) = 9.300961 + 0.0047026 x_3 x_5 + 0.0012547 x_1 x_3 + 0.0019085 x_3 x_4$$
$$0 \le g_1(x) \le 92$$
$$90 \le g_2(x) \le 110$$
$$20 \le g_3(x) \le 25$$
$$78 \le x_1 \le 102,\ 33 \le x_2 \le 45,\ 27 \le x_i \le 45\ (i = 3,4,5).$$

The optimum value is $f(x^*) = -30665.539$ at $x^* = (78,33,29.995256025682,45,36.775812905788)$

5. **g05**

Minimize $f(x) = 3x_1 + 0.000001 x_1^3 + 2x_2 + (0.000002/3) x_2^3$

subject to:

$$g_1(x) = -x_4 + x_3 - 0.55 \le 0$$
$$g_2(x) = -x_3 + x_4 - 0.55 \le 0$$
$$h_3(x) = 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0$$
$$h_4(x) = 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0$$
$$h_5(x) = 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0$$
$$0 \le x_i \le 1200\ (i = 1,2),\ -0.55 \le x_2 \le 0.55\ (i = 3,4).$$

The optimum value is $f(x^*) = 5126.4981$ at $x^* = (679.9463,1026.067,0.1188764,-0.3962336)$.

6. **g06**

Minimize $f(x) = (x_1 - 10)^3 + (x_2 - 20)^3$

Subject to:

$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \le 0$$
$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \le 0$$
$$13 \le x_1 \le 100,\ 0 \le x_2 \le 100$$

The optimum value is $f(x^*) = -6961.81388$ at $x^* = (14.095,0.84296)$.

Appendix A

(a) *F*$_1$: Optimal Capacity of Gas Production Facilities [20]

This is the problem of determining the optimum capacity of production facilities that combine to make an oxygen producing and storing system. Oxygen for basic oxygen furnace is produced at a steady state level. The demand for oxygen is cyclic with a period of one hour, which is too short to allow an adjustment of level of production to the demand. Hence the manager of the plant has two alternatives.

He can keep the production at the maximum demand level; excess production is lost in the atmosphere.

He can keep the production at lower level; excess production is compressed and stored for use during the high demand period.

The mathematical model of this problem is given by:

Minimize: $f(x) = 61.8 + 5.72x1 + 0.2623[(40 - x_1)\ln(\frac{x_2}{200})]^{-0.85} + 0.087(40 - x_1)\ln(\frac{x_2}{200}) + 700.23 x_2^{-0.75}$

Subject to: $x_1 \ge 17.5, x_2 \ge 200; 17.5 \le x_1 \le 40, 300 \le x_2 \le 600$

(b) F_2: Optimal Thermo hydraulic Performance of an Artificially Roughened Air Heater [21]

In this problem the optimal thermohydraulic performance of an artificially roughened solar air heater is considered. Optimization of the roughness and flow parameters (p/e, e/D, Re) is considered to maximize the heat transfer while keeping the friction losses to be minimum. The mathematical model of this problem is given by:

Maximize $L = 2.51 * \ln e^{+} + 5.5 - 0.1R_M - G_H$

where $R_M = 0.95x_2^{0.53}$; $G_H = 4.5(e^{+})^{0.28}(0.7)^{0.57}$

$e^{+} = x_1x_3(\bar{f}/2)^{1/2}$; $\bar{f} = (f_s + f_r)/2$ $f_s = 0.079x_3^{-0.25}$; $f_r = 2(0.95x_3^{0.53} + 2.5 * \ln(1/2x_1)^2 - 3.75)^{-2}$

Subject to: $0.02 \le x_1 \le 0.8, 10 \le x_2 \le 40, 3000 \le x_3 \le 20000$

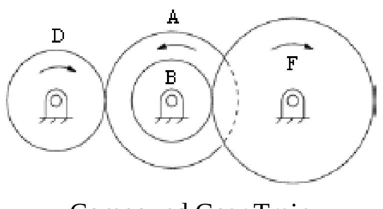
(c) F_3: Design of Gear Train [22]

This problem is to optimize the gear ratio for the compound gear train. This problem shown in Figure 5 was introduced by Sandgren [32]. It is to be designed such that the gear ratio is as close as possible to 1/6.931. For each gear the number of teeth must be between 12 and 60. Since the number of teeth is to be an integer, the variables must be integers. The mathematical model of gear train design is given by,

$$Minimize\ f = \left\{\frac{1}{6.931} - \frac{T_dT_b}{T_aT_f}\right\}^2 = \left\{\frac{1}{6.931} - \frac{x_1x_2}{x_3x_4}\right\}^2$$

Subject to: $12 \le x_i \le 60 \quad i = 1,2,3,4$

$[x_1, x_2, x_3, x_4] = [T_d, T_b, T_a, T_f]$, x_i's should be integers. T_a, T_b, T_d, and T_f are the number of teeth on gears A, B, D and F respectively. The design of a compound gear train is shown in Figure 2.



Compound Gear Train

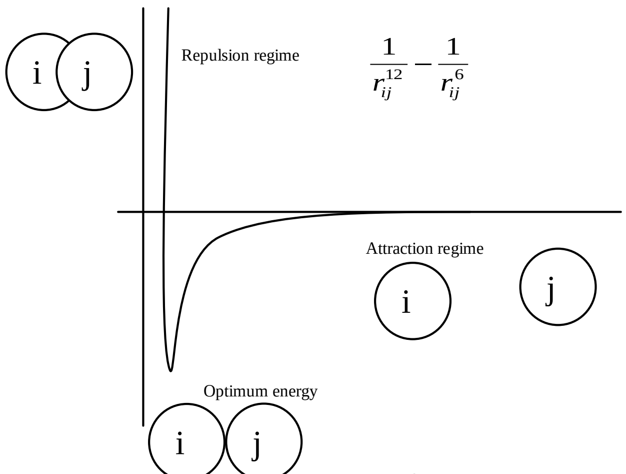
(d) F_4: The lennard-jones atomic cluster problem [23]

The Lennard-Jones (LJ) model of inert gas cluster has been investigated intensively as a challenging testing ground for global optimization algorithms. The LJ problem can be formulated as follows: Let N atoms be given in three dimensional space. Let $x^i = (x_1^i, x_2^i, x_3^i)^T$ represent the coordinates of atom *i*. let $X = ((x^1)^T, \ldots, (x^N)^T)^T$ the LJ potential energy function $v(r_{ij})$ of a pair of atoms (*i, j*) is given by $v(r_{ij}) = \frac{1}{r_{ij}^{12}} - \frac{1}{r_{ij}^{6}}, 1 \le i,j \le N \ \ where\ r_{ij} = ||x^i - x^j||$

The LJ potential function for a single pair of neutral atoms is a simple unimodal function. This is illustrated by Figure 3. It is easy to find the overall minimum of this function that is assumed at 1 with energy −1. In a complex system, many atoms interact and we need to sum up the LJ potential functions for each pair of atoms in a cluster. The result is a complex energy landscape with many local minima. It is given by:

$$Minimize\ V(X) = \sum_{i<j} v(\| x^i - x^j \|) = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\left(\frac{1}{\| x^i - x^j \|^{12}} - \frac{1}{\| x^i - x^j \|^{6}}\right)$$

As it is known the LJ problem is a highly nonlinear, non convex function with numerous local minimum that takes the LJ problem a challenging standard test problem for the global optimization algorithms. For illustration, we fix one atom's position and let the others move around the fixed one. A cluster of more than 20 atoms has many of local minima along its LJ PES.



LJ potential of a single pair of atoms

(e) F_5: Gas transmission compressor design [20]

Minimize $f(x) = 8.61*10^5 * x_1^{1/2} x_2 x_3^{-2/3} (x_2^2 - 1)^{-1/2} + 3.69*10^4 * x_3 + 7.72*10^8 * x_1^{-1} x_2^{0.219} - 765.43*10^6 * x_1^{-1}$

Bounds: $10 \le x_1 \le 55, 1.1 \le x_2 \le 2, 10 \le x_3 \le 40$

Tarun Kumar Sharma: Assistant Professor in Amity Institute of Information Technology, Amity University Rajasthan, India. He has an experience of 11 years (4 years of research and 7 years of Academics). His research areas are evolutionary algorithms and their applications in Software Engineering. He is in Editorial Board and reviewer of many refereed Journals. He has published about 40 research papers in Journal of repute and in refereed international Journals.

Millie Pant: Associate Professor, Indian Institute of Technology, Roorkee, India. She has published above 200 research publications in referred journals and international conferences. She has been keynote speakers to various seminars, conferences and development programs. Her key research areas are Evolutionary Computing, Swarm Intelligence and their application in various areas of Engineering.