The Bees Algorithm Technical Note

Article · September 2005

CITATIONS
202

READS
12,144

6 authors, including:

D. T. Pham
University of Birmingham
622 PUBLICATIONS **15,427** CITATIONS
SEE PROFILE

Afshin Ghanbarzadeh
Shahid Chamran University of Ahvaz
106 PUBLICATIONS **3,471** CITATIONS
SEE PROFILE

Ebubekir Koç
Fatih Sultan Mehmet Vakif Üniversitesi
63 PUBLICATIONS **2,369** CITATIONS
SEE PROFILE

Sameh Otri
Cardiff and Vale College
32 PUBLICATIONS **2,171** CITATIONS
SEE PROFILE

Technical Note

Bee Algorithm

A Novel Approach to Function Optimisation

D.T. Pham , A. Ghanbarzadeh , E. Koc , S. Otri , S. Rahim , M. Zaidi

The Manufacturing Engineering Centre
Cardiff University
Queen's University
The Parade
Newport Road
Cardiff CF24 3AA

Technical Note: MEC 0501 **Dec 2005**

Bee Algorithm

A Novel Approach to Function Optimisation

D.T. Pham , A. Ghanbarzadeh , E. Koc , S. Otri , S. Rahim , M. Zaidi

The Manufacturing Engineering Centre
Cardiff University
Queen's University
The Parade
Newport Road
Cardiff CF24 3AA

Technical Note: MEC 0501 **Dec 2005**

Table of Content

Technical Note MEC 0501

Bee Algorithm

A Novel Approach to Function Optimisation

D.T. Pham , A. Ghanbarzadeh , E. Koc , S. Otri , S. Rahim , M. Zaidi

The Manufacturing Engineering Centre
Cardiff University
Queen's University
The Parade
Newport Road
Cardiff CF24 3AA

Abstract

Researchers have been inspired by nature in many different ways. A new population based search method called the *Bee Algorithm (BA)* is proposed. The swarm based algorithm described in this paper goes beyond the traditional design and revolves around mimicking the food foraging behaviour of honey bees which has been applied to solve unconstrained functions in the domain of continuous space. The authors believe this algorithm is very efficient and robust, as indicated by the extensive simulation results on numerous benchmarking problems.

Keywords: Bee Algorithm, Continuous Function Optimization, Swarm Intelligence, Search Strategies

1. Introduction

Researchers have been inspired by nature in many different ways. Classical optimization methods encounter great difficulty when faced with the challenge of solving hard problems with acceptable levels of time and precision. It is generally believed that NP-hard problems cannot be solved to optimality within polynomially bounded computation times, thus generating much interest in approximation algorithms that find near-optimal solutions within reasonable running times. The swarm based algorithm described in this paper goes beyond the traditional design and revolves around mimicking the food foraging behaviour of scout bees.

Section 2 details the literature review and previous works related to bees. Section 3 describes the foraging behaviour of natural bees as well as the core idea of our Bee Algorithm. In section 4, the simulation work is discussed. Section 5 details the results and benchmarking problems which were used to test the robustness and agility of the bee algorithm. Section 6 gives the concluding remarks and further discussions.

2. Literature Review and Related Works

A recent trend is the introduction of non-classical stochastic search optimization algorithms. The swarm based evolutionary algorithms (EA) mimic nature's evolutionary principles to drive their search towards an optimal solution. One of the most striking differences from direct search algorithms such as hill climbing and random walk is that EAs use a population of solutions for every iteration, instead of a single solution. Since a population of solutions are processed in every iteration, the outcome is also a population of solutions. If an optimization problem has a single optimum, all EA population members can be expected to converge to that optimum solution. However, if an optimization problem has multiple optimal solutions, an EA can be used to capture multiple optimal solutions in its final population. These approaches include Ant Colony Optimization (ACO) [4], Genetic Algorithm (GA) [3] and Particle Swarm Optimization (PSO) [5].

Common to all population based search methods is a strategy that generates variations of the tuning parameters. Most search methods use a greedy criterion to make this decision, which accepts the new parameter if and only if it reduces the value of the objective or cost function. One of the most successful metaheuristic is the ACO, which is built on real ant colony behaviour. Ants are capable of finding the shortest path from the food source to their nest using a chemical substance called pheromone. The pheromone is deposited on the ground as they walk and the probability that a passing stray ant will follow this trail depends on the quantity of pheromone laid. ACO was

first used for continuous optimization by Bilchev [1] and further attempts were made in [1,2].

The Genetic Algorithm is based on artificial selection and genetic recombination. This works by selecting two parents from the current population and then by applying genetic operators – mutation and crossover – to create a new gene pool set. They efficiently exploit historical information to speculate on new search areas with improved performance [3]. When applied to optimization problems, GA has the advantage of performing global search and may hybridize with domain-dependant heuristics for better results. [2] describes such a hybrid algorithm of ACO with GA for continuous function optimization.

The idea of honey bees has been used inadequately in different topics – mostly in discrete space. BeeHive, BeeAdHoc and Bee Algorithm [8, 9, 6] are all algorithms inspired from the bee swarming behaviour. A model generated by studying the allocation of bees to different flower patches to maximize the nectar intake is described in [6]. This was subsequently applied to distribute web applications at hosting centres. Another model borrowing from the principles of bee communication is presented in [8]. The artificial bee agents are used in packet switching networks to find suitable paths between nodes by updating the routing table. Two types of agents are used – short distance bee agents which disseminate routing information by travelling within a restricted number of hops and long distance bee agents which travel to all nodes of the network. Though the paper talks in terms of bees, it only loosely follows their natural behaviour.

In [7], the author describes a virtual bee algorithm where the objective function is transformed into virtual food. The paper is like a black box in that no information about how the transformation from objective function to food source or agent communication is highlighted. Nor are there any comparative results to check the validity of the algorithm.

3. The Bee Algorithm

3.1. Natural Bees

A colony of honey bees can be explained as a large and diffuse creature which can extend itself over great distances (more than 10 km) and in multiple directions simultaneously to exploit a vast number of food sources.[12,13] A colony can only be successful by deploying its foragers to good fields. In principle, flower patches that provide plentiful amounts of nectar or pollen that are easy to collect with less energy usage, should receive more bees, whereas, patches with less nectar or pollen should receive fewer bees. [10, 11]

Similar to other social insect colonies, honey bees also have the same constituents for self-organization as classified in [10]. Self organization relies on four basic ingredients: (1) Positive feedback, (2) negative feedback, (3) the amplification of fluctuations – random walks, errors- and (4) multiple interactions. In bee colony, the objective is to increase the amount of honey by bringing more nectar, pollen and water with good enough quality. Like other insects, achievement is not easy task in a short term harvesting season for that sized animals. Thus, self-organization has a vital role in the colony. For positive feedback bees have a special dancing behaviour which was discovered by Karl von Frisch. [12] This waggle dance is essential for exploitation of good sources, so it has a vital role in colony. Negative feedback is another issue which is balancing the positive feedback effect in colony. During harvesting when the bees detect that there is no more nectar appears in a flower patch bees will be abandoned and they follow other dances on dance floor. Randomness is also one of the essential

part of the bee colony, especially in searching process. Although, bees have some senses to decrease the chance factor they still follow the basic random search idea. And individuals use others information that presented by a dance, as a multiple interaction part.[10,11,12,13]

The foraging process begins in a colony by sending scout bees to search for adequate flower patches. Scout bees search randomly through their journey from one patch to another. Moreover, during the whole harvesting season, a colony continues its exploration, keeping a percentage of the whole population as scout bees. [13]

When they return to the hive, those who found a source which is above a certain threshold (a combination of some constituents, such as sugar percentage regarding the source) deposit their nectar or pollen and go to the dance floor to perform their waggle dance. [12]

This mysterious dance is essential for colony communication, and contains three vital pieces of information regarding flower patches: direction, distance, and quality of source. [11, 12] This information guides the colony to send its bees to flower patches precisely, without any supervisory leader or blueprint. Each individual's knowledge of the outside environment is gleaned solely from the waggle dance. This dance gives the colony a chance to evaluate different patches simultaneously in addition to minimising the energy usage rate.[11] After waggle dancing on the dance floor, the dancer bee (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. This allows the colony to gather food quickly and efficiently.

While harvesting the source, the bees monitor the food level, which is necessary to decide upon the next waggle dance when they return to the hive. [11] If the food

source is still good enough and calls for more recruitment, then that patch will be advertised by making a waggle dance and recruiting more bees to that source.

3.2. The Proposed Bee Algorithm

The Bee Algorithm is inspired by the natural foraging behaviour of honey bees to find the optimal solution. Fig 1 shows the pseudo code for a simple Bee Algorithm.

The algorithm starts by initialising a set of parameters namely the: number of scout bees (n), number of elite bees (e), number of selected regions out of n points (m), number of recruited bees around elite regions, number of recruited bees around other selected (m-e) regions, and stopping criteria Then, N bees are placed on the search space randomly, similar to scout bees. Every bee on the problem space evaluates the fitness of its field in step 2. Similar to the natural searching (or scouting) process carried out by scout bees.

Subsequently, in step 4, elite bees that have better fitness are selected and saved for the next population. Then, in neighbourhood search (step 5 to 7), the algorithm starts by assigning elite bees positions to search directly around them and other positions will be chosen, either the best ones or stochastically using a roulette wheel proportional to fitness. In step 6, the bees search around these points within the boundaries and their individual fitness is evaluated. More bees will be recruited around elite points and fewer bees will be recruited around the remaining selected points. Recruitment is one of the essential hypotheses of the bee algorithm along with the scout bee concept, both of which are used in nature.

1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criteria not met) //Forming new population.
4. Select elite bees.
5. Select sites for neighbourhood search.
6. Recruit bees around selected sites and evaluate fitness.
7. Select the fittest bee from each site.
8. Assign remaining bees to search randomly and evaluate their fitness.
9. End While

Fig.1. Pseudo code of the bee algorithm

However, in step 7, for each site only one representative bee with the highest fitness will be selected. In nature there is no such restriction. Bees behave more flexibly, but here in term of optimisation and efficiency it is enough to select only one representative from each site. In step 8, the remaining bees will be sent randomly around the search space. Similar to honey bees, the algorithm keeps its random searching part by sending scout bees to find new potential solutions. These steps will be repeated until certain criterion is met.

At the end of each iteration, the colony will organise itself by having three parts to its new population - elite bees, representatives from each selected site, and remaining bees assigned to random search.

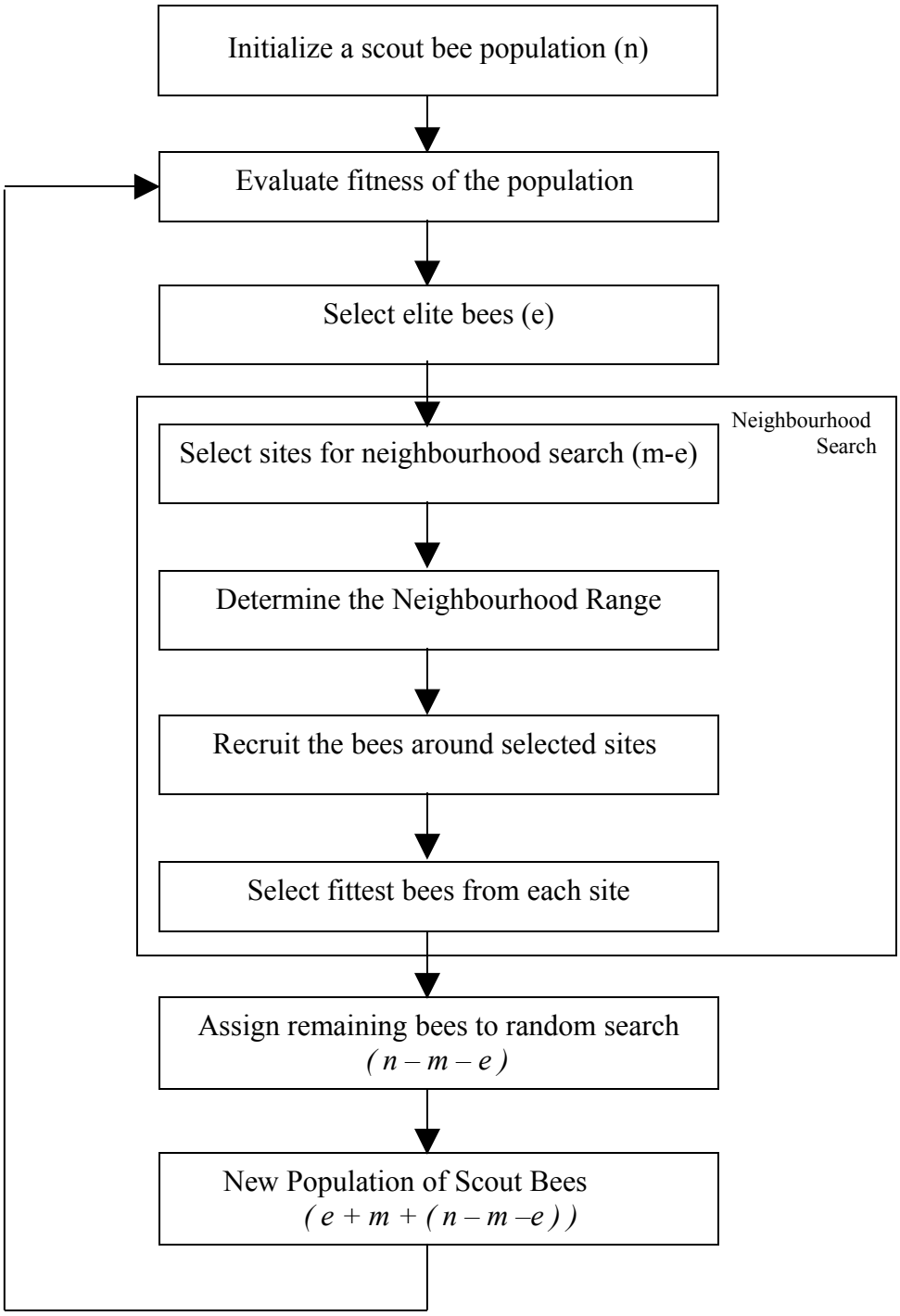
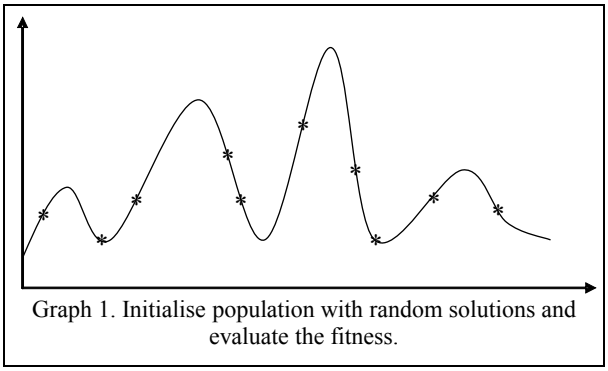
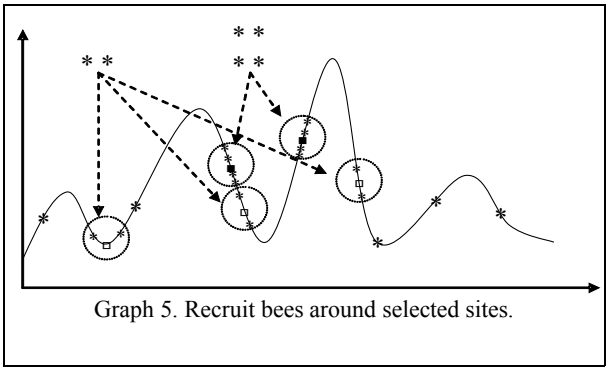


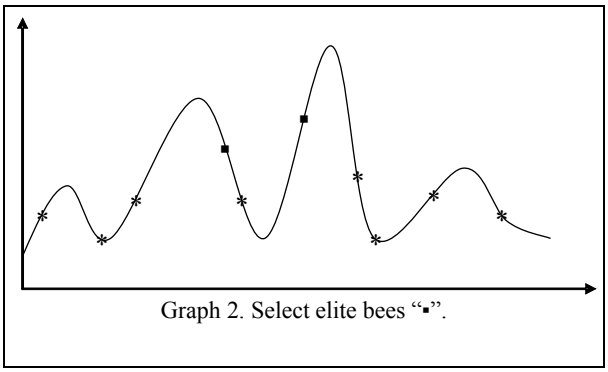
Fig.2. Flowchart of basic bee algorithm



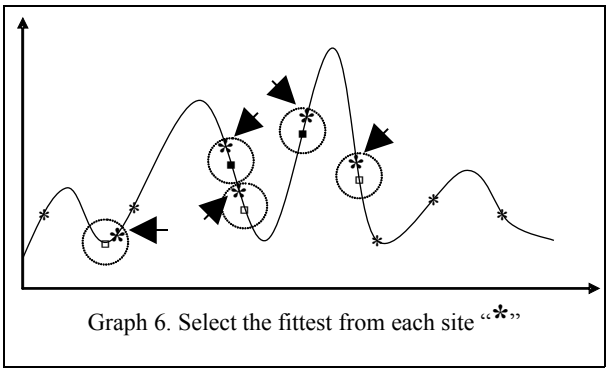
Graph 1. Initialise population with random solutions and evaluate the fitness.



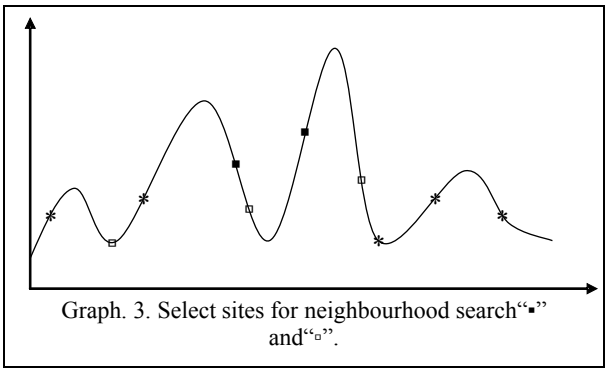
Graph 5. Recruit bees around selected sites.



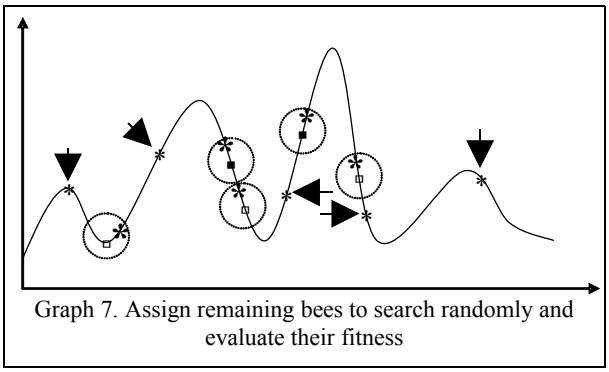
Graph 2. Select elite bees "▪".



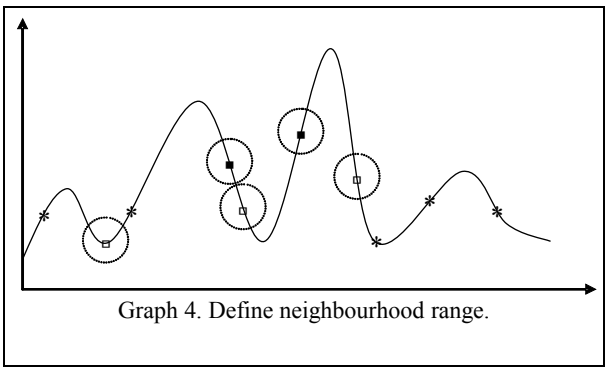
Graph 6. Select the fittest from each site "✻"



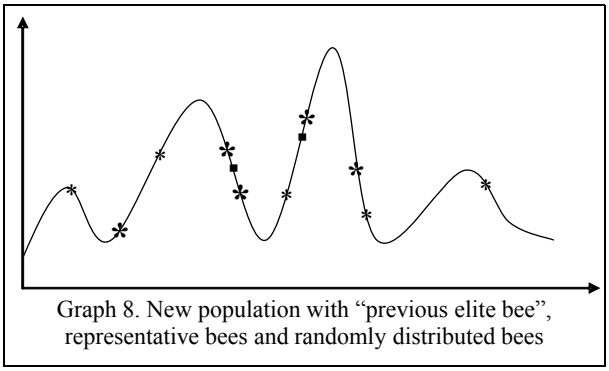
Graph. 3. Select sites for neighbourhood search"▪" and"▫".



Graph 7. Assign remaining bees to search randomly and evaluate their fitness



Graph 4. Define neighbourhood range.



Graph 8. New population with "previous elite bee", representative bees and randomly distributed bees

Fig.3. Simple example of bee algorithm.

3.3. A Study of Some Characteristics of Bee Algorithm

In this section different parameters of algorithm and their effect will be discussed.

3.3.1. Population Size of the Bee Algorithm

Population (n) is one of the key parameters in the Bee Algorithm. Effect of changing population size on the mean number of iteration to get the answer, number of evaluated points and reliability of successfulness of the algorithm are shown in the figures 4, 5 and 6.

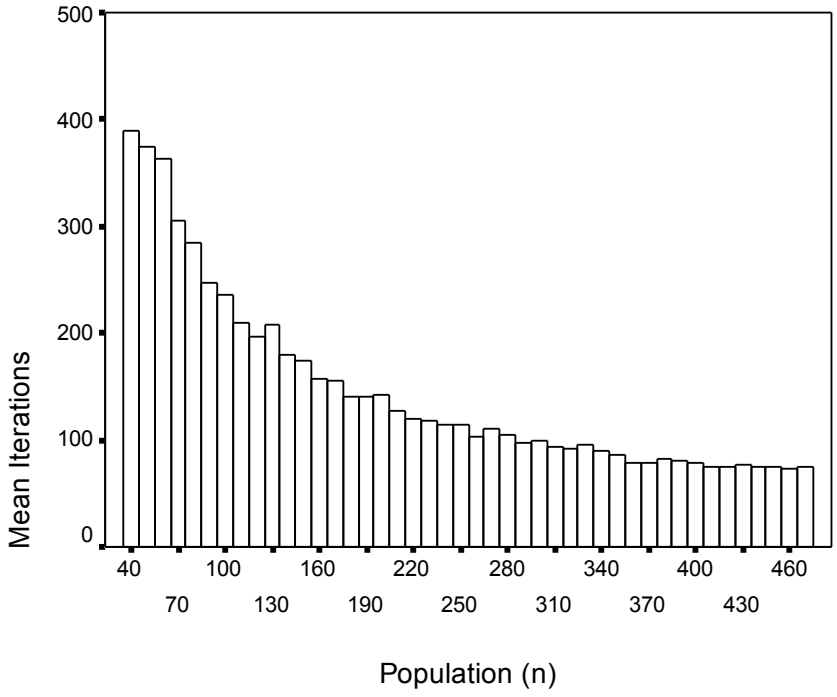


Fig.4. Population size versus Mean iteration

In fig.4, with increasing the population size the iteration required to obtain the answer will reduce.

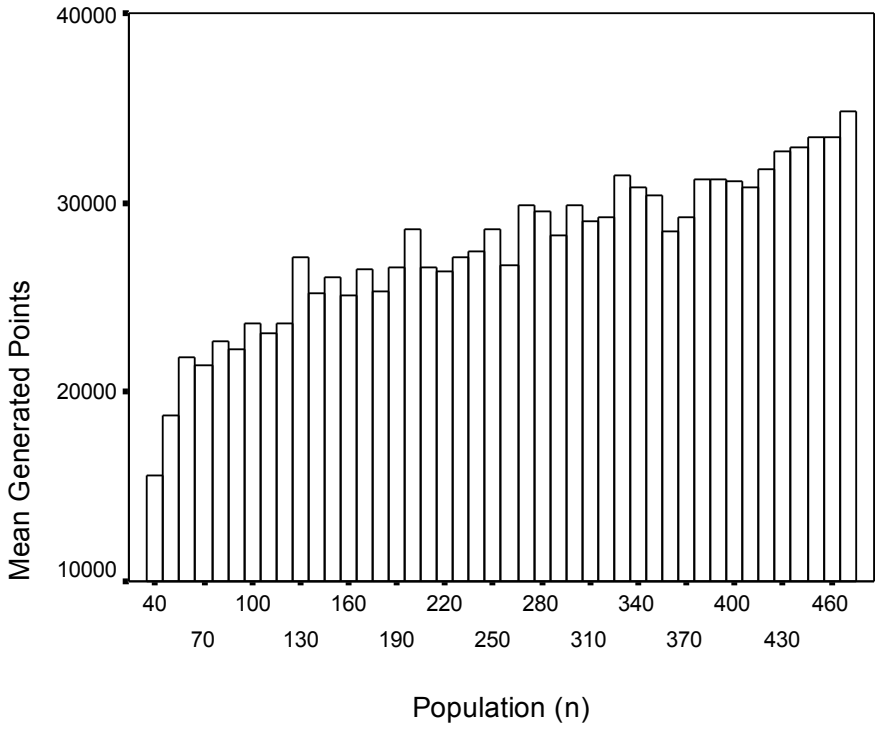


Fig.5. Population size versus Number of function evaluation

According to fig.5, with the increase of population size, number of function evaluation will increase which is predictable.

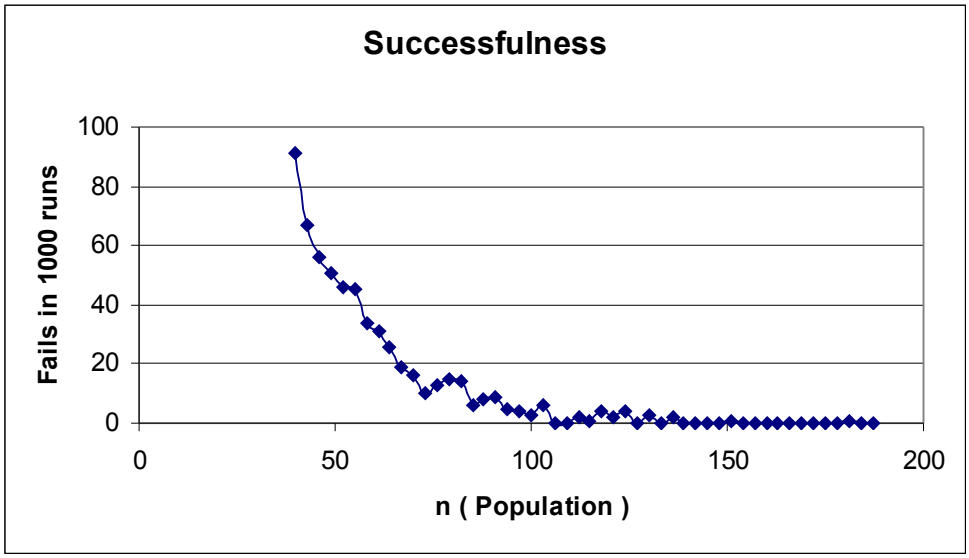


Fig.6 Successfulness for different population size

To have a good reliability, a minimum size of population is required (Fig. 6). In order to have less number of iteration to get the answer, population should be large, to get a reasonable number of function evaluations, population has to be as less as possible and for have a good reliability population should be more than a size. These three constitutions will give a good range to choose the size of population.

3.3.2. Elitism

Number of elites as long as it is more than one, does not have major effect on the performance of the Bee Algorithm, and thus the number of elites can be a small number more than zero.

3.3.3. Neighbourhood Search

Neighbourhood search is an essential concept for all evolutionary algorithms as well as the bee algorithm. In the bee algorithm, searching process in neighbourhood range is similar to foraging field exploitation of natural bees. As explained above, when a scout bee find any good enough foraging field, she advertise it back into the hive for recruiting more bees to that field. This behaviour is helpful to bring more nectar into the hive. Hence, this fruitful method might be also helpful for engineering optimization problems.

Harvesting process also includes a monitoring process which is necessary for decision making for a waggle dance back into the hive to recruit more bees to that site. In the bee algorithm, this monitoring process can be used as neighbourhood search. Principally, when a scout bee finds a good field (good solution), she advertise her field to more bees. Subsequently, those bees fly to that source take piece of nectar and turn back to hive again. Depending on the quality, source can be advertised by some of the bees that already know the source. In proposed bee algorithm, this behaviour has been used as a neighbourhood search. As explained above, from each foraging site (or neighbourhood site) only one bee is chosen. This bee must have the best solution information about that field. Thus, algorithm can create some solutions which related to previous ones.

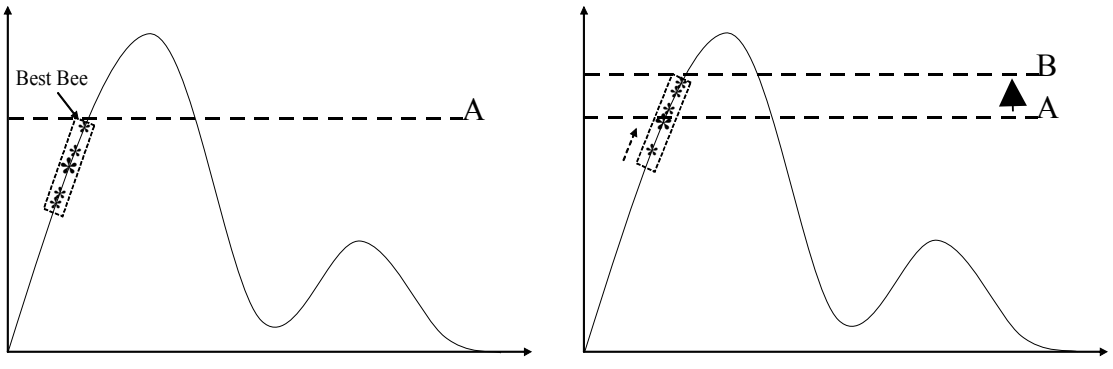


Fig.7. Graphical explanation of neighbourhood search

However, neighbourhood search based on random distribution of bees in a predefined neighbourhood range. For every selected site, bees are randomly distributed to find a better solution. As shown in fig. 7, only the best bee is chosen to advertise her source and the centre of the neighbourhood field shifted to best bees` position (from A to B).

Moreover, during harvesting process other constituents must also be taken into account for increasing the efficiency; number of recruited bees in neighbourhood range and range width.

Number of recruited bees around selected sites, should be defined properly. When the number is increased then the number of function evaluation will also be increased. Vice verse, when the number decreases, chance to find a good solution will also be decreased.

This problem also depends on the neighbourhood range. If range can be arranged adequately, then number of recruited bees will depend on the complexity of a solution space. This will be discussed later with more details.

3.3.3.1. Site Selection

Two deferent techniques have been implemented: Probabilistic Selection and Best Site Selection.

In the probabilistic selection, roulette wheel method has been used and sites with better fitness have more chance to be selected, but in the best selection, the best sites according to fitness will be selected. Of course in this report different combinations of selection of two methods from pure probabilistic selection (q=0) to pure best selection (q=1) have been investigated and mean iterations required to rich the answer and successfulness are shown in Fig.8. and 9.

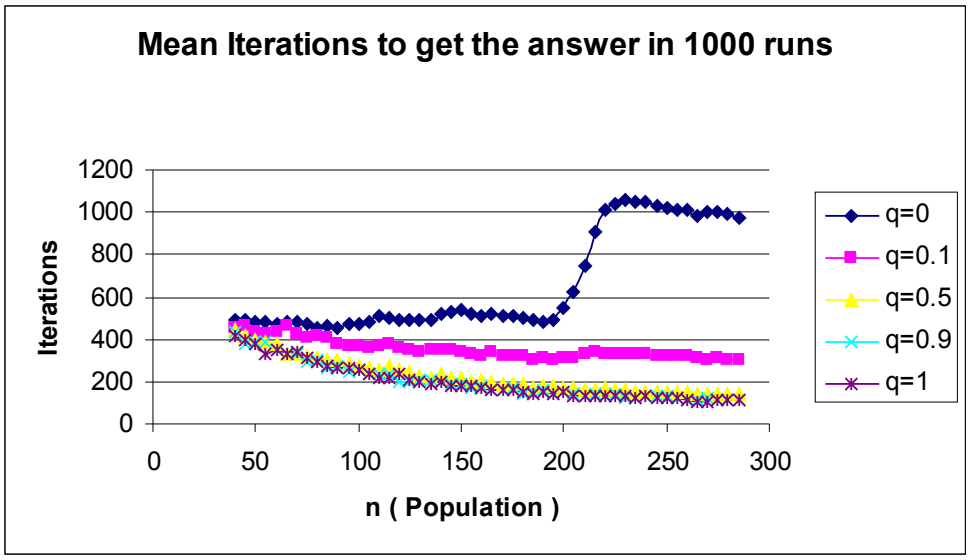


Fig.8. Mean iteration required for Different combination of selection

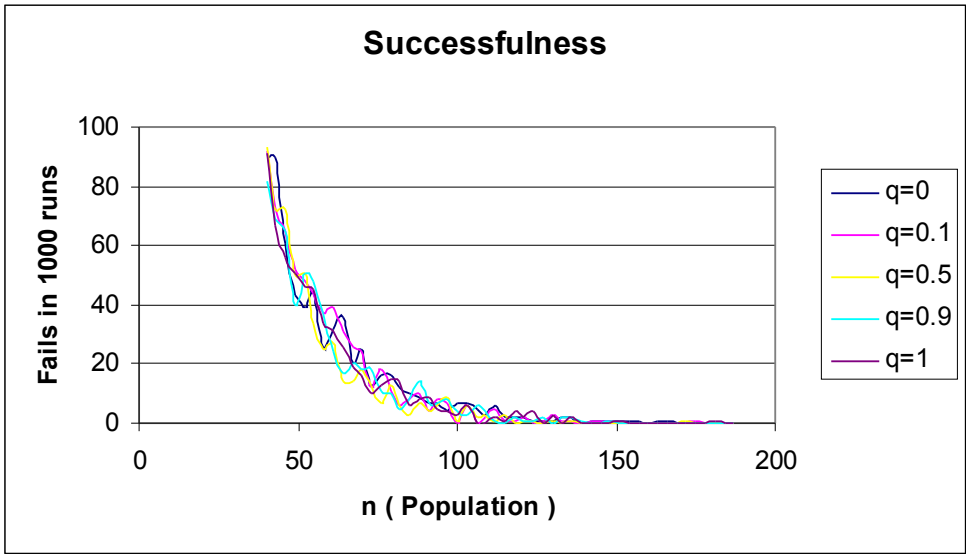


Fig.9. Successfulness of different combinations of selection methods

Regarding to results, best selection will present better results and also is more simple, so Best Selection technique is recommended to use.

3.3.3.2. Neighbourhood Range

Neighbourhood range is another variable which needs to tune for different types of problem spaces. And also, it is necessary to discuss different strategies for increasing the robustness and quality of the algorithm. In this section, different strategies have been discussed.

Four different strategies have been applied to improve the efficiency and robustness of the bee algorithm. These are (1) Fixed neighbourhood region width strategy, (2) region changing according to iteration and (3) hybrid strategies between previous two

strategies, for instance, first strategy has been used up to 50th iteration then second strategy have been used up to end (or up to optimum point).

In first strategy, for all selected sites neighbourhood width has been fixed to a certain range that is enough to deal with the complexity of a problem space. In this strategy, beginning from the first iteration all the bees harvest on the same size of the fields which is defined as "α" in equation (1).

$$Neighbourhood\_Range(2ndStrategy) = \alpha; \quad (1)$$

However, in second strategy, region changes proportional to iteration. All sites have the same range value and this range will be narrowed down depends on iteration as shown in equation (2). This strategy has been established on an idea to increase the accuracy of the solution.

$$Neighbourhood\_Range(3rdStrategy) = \frac{\alpha}{\text{iteration}}; \quad (2)$$

Finally, a third strategy has been implemented as a combination of first two strategies. This model has been implemented to improve the efficiency of prior strategies.

Neighbourhood range strategies have been tested using two functions that have also been used in benchmarking experiments presented in table 1. These tests run independently 10 times and mean of this 10 run was presented on Fig. 9. Also number of iteration set up to both tests as 1000 iterations.

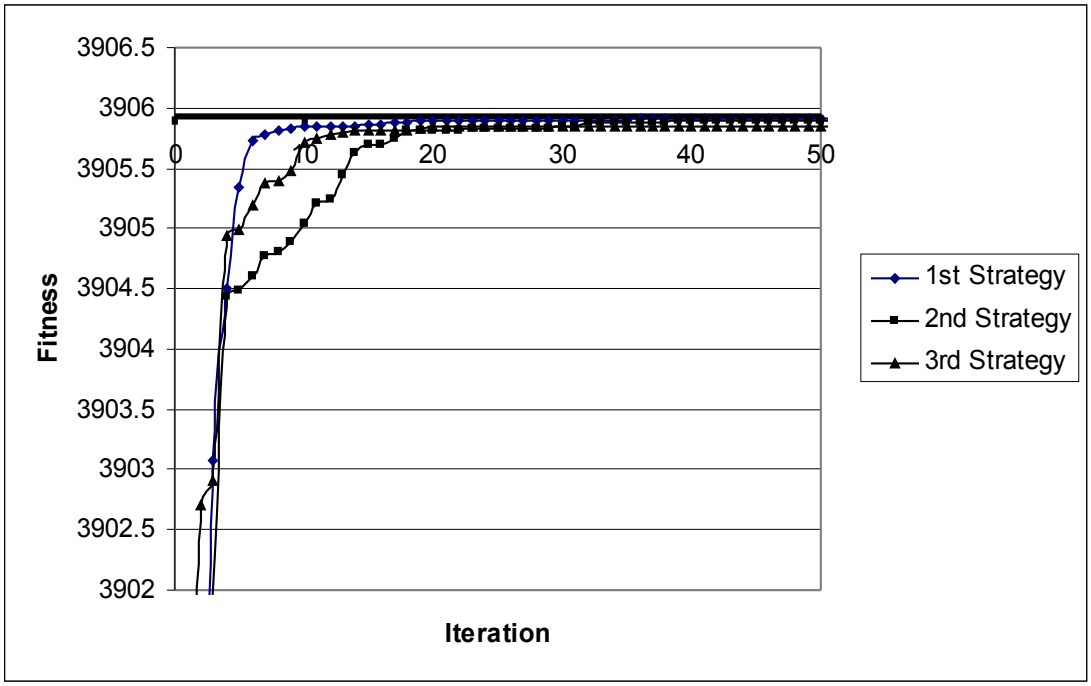


Fig.10. Comparison of different neighbourhood strategies for De Jong Function.

In Fig. 10, three different strategies have been experimented on De Jong test function. The following parameters were set for this test.- population is 15, selected sites is 5, elite site is 2, bees around elite points are 4, bees around selected points are 2. However, neighbourhood spread 0.01 is defined as a initial range for all strategies. Also for first strategy, first 20 iteration algorithm will follow the first strategy (fixed ranges) and after that follows second strategy (narrowing down ranges).

Clearly, first strategy reaches the minimum before then other strategies. But, second and third strategies are also gives similar results for this relatively simple problem space as well as first strategy. Thus, it does not make sense to use a complex method like second and third methods. So, the first strategy is a simple and an efficient method for neighbourhood search

3.3.3.3. Recruitment

Recruitment can be done by different methods, which in this research recruitment proportion to the fitness of the selected sites and equal distribution of bees on the selected sites have investigated, which for simplicity the second method, equal distribution, has been chosen. Although numbers of recruited bees are fixing for each site, but the elite sites will recruits more bees (nep) and other selected sites recruit less amount of bees (nsp).

4. Simulation and Comparison

Two standard functions problems were selected to test the bee algorithm and another ten for benchmarking. Since the Bee Algorithm searches for the maximum, the function is inverted for minimization problems.

Shekel's Foxholes (Fig. 11), a 2D equation from De Jong's test suite (function 5) is chosen as the first function for testing the algorithm (Eq. 3.).

$$\frac{1}{f_5(\vec{x})} = 119.998 - \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_{i-} a_{ij})^6}$$

(3)

$$a_{ij} = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & \dots & 32 & 32 & 32 \end{pmatrix}$$

$$-65.536 \le X_i \le 65.536$$

$$\max(f_5) = (f_5)(-32,-32) \approx 119$$

The following parameters were set for this test.- population n = 45, selected sites m = 3, elite site e = 1, neighbourhood spread ngh = 0.6, bees round elite points nep = 7, bees around selected points nsp = 2.

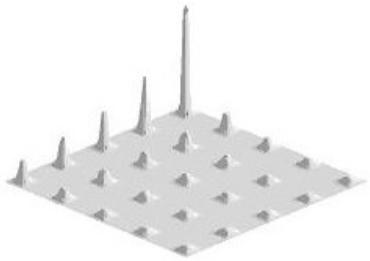


Fig.11. Inverted Shekel's Foxholes

Fig.12 shows the number of function evaluations to average fitness for 100 independent runs. It can be seen that after approximately 1200 function evaluations, the Bee Algorithm reaches close proximity of the optimum.

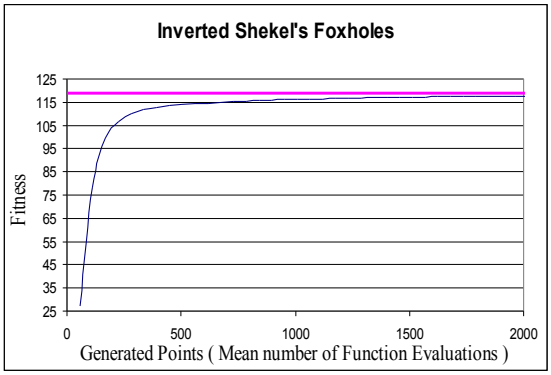


Fig.12. Result for fitness average and Generated points

To test the reliability of the algorithm, Inverted Schwefel's function with six dimensions was chosen (Eq.4). In Fig. 13, a two dimensional model is shown to highlight the complexity of this function.

$$f(x) = -\sum_{i=1}^{6} -x_i \sin(\sqrt{|x_i|}) \tag{4}$$

$$-500 \le X_i \le 500$$

$$\max f(x) = 6 * 418.9829; X_i = 420.9687, i = 1:6$$

The following parameters were set for this test.- population n = 500, selected sites m=15, elite site e = 5, neighbourhood spread ngh = 0.01, bees around elite points nep= 50, bees around selected points nsp = 30.

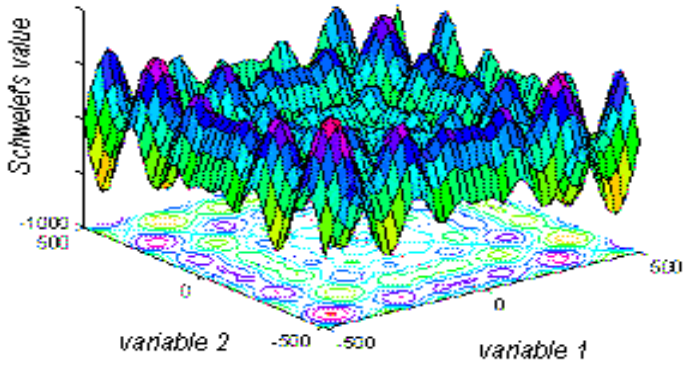


Fig.13. 2D Schwefel's function

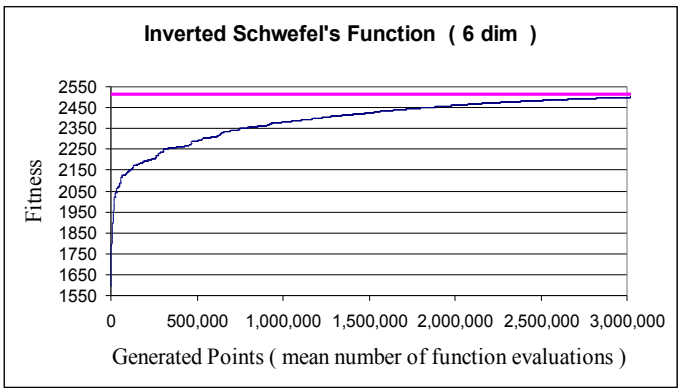


Fig.14. Generated points to Avg. fitness

Fig. 14 illustrates the function evaluations versus average fitness for 100 independent runs. It can be seen that after approximately 3,000,000 function evaluations, the bee algorithm reaches close to the optimum.

To compare the Bee Algorithm with other methods, eight functions [2] of various dimensions were chosen to test the robustness and speed. Test functions, their intervals, and the global optimum for each have been described in Table 1.

No	Reference	Interval	Test Function	Global Optimum
1	De Jong	X[-2.048,2.048]	$\max F = (3905.93) - 100(x_1^2 - x$	X[1,1] F=3905.93
2	Goldstein & Price	X[-2,2]	$\min F = [1+(x_1+x_2+1)^2(19-$ $X[30+(2x_1-3x_2)^2(18-32x$	X[0,-1] F=3
3	Branin	X[-5,10]	$\min F = a(x_2 - b x_1^2 + c x_1 - d)^2 +$ $a=1, b=\frac{5.1}{4}\left(\frac{7}{22}\right)^2, c=\frac{5}{22}X7, d$	X[-22/7,12.275] X[22/7,2.275] X[66/7,2.475] F=0.3977272
4	Martin & Gaddy	X[0,10]	$\min F = (x_1 - x_2)^2 + ((x_1 + x_2 -$	X[5,5] F=0
5	Rosenbrock -1	X[-1.2,1.2] X[-10,10]	$\min F = 100(x_1^2 - x_2)^2 + (1 - x_1$	X[1,1] F=0
6	Rosenbrock - 2	X[-1.2,1.2]	$\min F : \sum_{i=1}^{3} \{100(x_i^2 \cdot x_{i+1})^2 + (1 \cdot x_1)^2\}$	X[1,1,1,1] F=0
7	Hyper sphere model	X[-5.12,5.12]	$\min F = \sum_{i=1}^{6} x_i^2$	X[0,0,0,0,0,0] F=0
8	Griewangk	X[-512,512]	$\max F = \frac{1}{0.1 + \left(\sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos\left($	X[0,0,0,0,0,0,0,0, 0,0] F=10

func no	SIMPSA		NE SIMPSA		GA		ANT		Bee Algorithm	
	success %	mean no of func evals	success %	mean no of func evals	success %	mean no of func evals	success %	mean no of func evals	success %	mean no of func evals
1	***	*****	***	*****	100	10160	100	6000	100	**49**
2	***	*****	***	*****	100	5662	100	5330	100	**998.9**
3	***	*****	***	*****	100	7325	100	1936	100	**1657.4**
4	***	*****	***	*****	100	2844	100	1688	100	**525.76**
5a	100	10780	100	4508	100	10212	100	6842	100	**898**
5b	100	12500	100	5007	***	*****	100	7505	100	**2306**
6	99	21177	94	3053	***	*****	100	8471	100	**29185**
7	***	*****	***	*****	100	15468	100	22050	100	**7112.9**
8	***	*****	***	*****	100	200000	100	50000	100	**1846.8**

Table 1. Test Functions.

Table 2. Bee Algorithms` comparison results between different algorithms

In Table 2. a comparison of the eight functions examined by bee algorithm with deterministic simplex method and the stochastic simulated annealing optimisation

procedure (SIMPSA and NE SIMPSA), genetic algorithm (GA) and ant colony approach (ANT)is shown. The mean number of function evaluations was obtained from 100 number independent runs and compared with other available results.

The first test function was De Jong, for which the Bee Algorithm could find the optimum 120 times faster than ANTS and 207 times faster than GA, with convergence for all them of 100%. The second function was Goldstein and Price, for which the Bee Algorithm reaches the optimum almost 5 times faster than ANT and GA with 100% reliability for all three methods. In the Branin function, there are 15% improvement compares with ANT and 77% over GA, again with 100% convergence.

Function 5 and 6, were Rosenbrock functions with two and four dimensions. In two dimensions Rosenbrock function in different intervals, the Bee Algorithm delivers a very good performance with 100% reliability and good improvement over other methods (at least two times less than other methods). In case of four dimensions, the Bee Algorithm needed more function evaluations to get the optimum with 100% success. NE SIMPSA could find the optimum in 10 times fewer function evaluations but the success was 94% and ANT method found the optimum with 100% success and 3.5 times faster than Bee Algorithm. Test function 7 was Hyper Sphere model with six dimensions which bee algorithm needed half of the function evaluations compares with GA and one third of those required for ANT method as reliability was 100%. Last test function was a ten dimensional problem known as Griewangk and Bee Algorithm shows huge differences with GA and ANT. Bee Algorithm could reach the optimum 10 times faster than GA and 25 times faster than ANT with 100% success.

5. Conclusion and Discussion

This paper presented a new algorithm for continuous function optimization. Experimental results on uni-modal and multi-modal functions in n-dimensions showed that the proposed algorithm has remarkable robustness and agility producing a 100% success rate. It converges to the maximum or minimum without becoming stuck at local optima. The Bee Algorithm outperformed other techniques that were compared in terms of speed and accuracy of the results obtained. One of the drawbacks of the algorithm would be the number of tuneable parameters used. However, it was possible to set the parameter values with a few trials.

Other evolutionary algorithms converge quickly making use of gradient information. However, our algorithm makes little use of this gradient field information in the neighbourhood, thus helping to escape from being stuck at local optima. Further work should address the reduction of tuneable parameters and the incorporation of better learning mechanisms.

Acknowledgement

The authors would like to thank MEC to provide the facility to do the research.

Appendix A **Program List**

Main :

```
// Bee2D.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <conio.h>
#include <iostream.h>
#include <limits.h>
#include <fstream.h>
#include <iomanip.h>
#include "params.h"
#include "func.h"

void main()
{
        cout<<"Started.....\n";
        int  i,d,j,k,temp2,aa[50],ranSearchBees,counter1,runs,fail,iter,loop1;  //50  max
number of selected sites out of n

        double temp1 ;
        double nb,sum_fit;
        double randselection;
        int a=20;
        int gen[1000],ite[1000]; // Generated Points
        int R=100; // number of runs
        double nghx[dim]; // Neighborhood X[]-Direction ( m )
        double q;      // random number which indicates Exploration or Exploitation


    int Flag_a = 1;    // 1: Equal distribution of bees in selected sites and more in ellit
sites
                       // 2: Proportion to fitness distribution of bees in selected sites

double bPos[dim][pop],bNghPos[dim][pop],fit[pop],bNghFit[pop],sortedFit[pop];
double percentsort[pop],candidx[dim][pop];
double totalsort = 0.0,shiftSortedFit[pop],sumsort[pop],bPosSort[dim][pop];
double gen_fit[700][100],mgen_fit[1000];

        ofstream Result;
        Result.open("Result.txt");
```

```
// Different random numer each time

srand( (unsigned)time( NULL ) );

for(runs=0; runs<R; runs++) {          //number of runs

        //Random distribution
        for(i=0;i<n;i++)
        {
                for(d=0;d<dim;d++)
                        bPos[d][i]=randfunc(start_x[d],end_x[d]) ;
                fit[i]=func(bPos,i);
        }
        ranSearchBees=n-m-e; // Number of bees search randomly

        //imax number of iteration
        for(iter=0; iter<imax ;iter++)
        {
                for(d=0;d<dim;d++)
                  nghx[d] = ngh;

        //sort fit & pos
        funcSort(fit, sortedFit, bPos, bPosSort, n);

          counter1=0;
          for(k=0;k<e;k++)
                  {
                   for(d=0;d<dim;d++)
                     bPos[d][counter1]=bPosSort[d][k];
                   counter1++;
                  }

          //Roulette wheel selection to choose sites or choosing best sites

          q=myrandom(); // choose a number b/w 0 and 1

          for (i=0;i<=n;i++) sumsort[i] = 0.0;

          if (sortedFit[n-1]<0.0)
                    for(k=0;k<n;k++)
                              shiftSortedFit[k]= sortedFit[k]-sortedFit[n-1]+1;
           else
                    for(k=0;k<n;k++)
                              shiftSortedFit[k] = sortedFit[k];

                  if(q < q0)        {         // Choosing best m
                            for(i=0;i<m;i++)
                                      for(d=0;d<dim;d++)
                                                candidx[d][i] = bPosSort[d][i];
```

```
                    }
                    else    {                       // Choosing sites probabalistically
                            for (i=0;i<m;i++)
                              {
                                        if( i < e ) // choosing the best (e) sites
                                        {
                                                for(d=0;d<dim;d++)
                                                        candidx[d][i]       =
bPosSort[d][i];
                                                shiftSortedFit[i]=0.0;
                                                continue;
                                        }
                                        // choosing the (m-e) sites using ( R/W)
                                        randselection = myrandom();

                                        totalsort = 0.0;

                                        for     (k=0;k<n;k++)     totalsort     +=
shiftSortedFit[k];

                                        for    (k=0;k<n;k++)    percentsort[k]    =
shiftSortedFit[k]/totalsort;

                                        for (k=1;k<=n;k++)
                                                sumsort[k]   =   sumsort[k-1]   +
percentsort[k-1];

                                        for (temp2=0;temp2<n;temp2++)
                                        {
                                           if  (sumsort[temp2]<randselection  &&
sumsort[temp2+1]>=randselection)
                                            {

                                            for(d=0;d<dim;d++)
                                                    candidx[d][i]    =    bPosSort[d]
[temp2];
                                            shiftSortedFit[temp2]=0.0;
                                            break;
                                            }
                                        }
                                }
                        }

                ////////////////////////////

                //    changing number of bees around each selected point

                    //condition for max values of n1 & n2
                    if (n1>(n-m-e)) n1 = n-m-e;
```

```
                    if (n2>(n-m-e)) n2 = n-m-e;

        if(Flag_a == 1)   //   Equal distribution of bees in selected sites
                    {
                            for(i=0;i<m;i++)
                            {
                             if(i<e)
                                      aa[i]=n2;  // Number of bees around each ellite point/s
                             else
                                       aa[i]=n1;   // Number of bees around other selected
point
                             }
                    }

                    else  //   Proportion to fitness distribution of bees in selected sites
                    {
                            sum_fit=0;

                            if (sortedFit[n-1]<0.0)
                            {
                                    for(i=0;i<m;i++)
                                             sum_fit=sum_fit+func(candidx,i)-sortedFit[n-1];

                                    for(i=0;i<m;i++)
                                     {
                                                                  nb=((func(candidx,i)-sortedFit[n-
1])/sum_fit)*a+.5;
                                                aa[i]=nb;    // Number of bees around each
selected point
                                                if(aa[i] < 1)
                                                aa[i]=1;
                                               }
                                    }
                             else
                             {
                                     for(i=0;i<m;i++)
                                             sum_fit=sum_fit+func(candidx,i);

                                     for(i=0;i<m;i++)
                                     {
                                              nb=(func(candidx,i)/sum_fit)*a+.5;
                                                aa[i]=nb;    // Number of bees around each
selected point
                                    if(aa[i]<1) aa[i]=1;
                                      }
                                     }
                            }


                    // Search in the neighbourhood
```

```
                temp1=-INT_MAX;

                for(k=0;k<m;k++)//k site
                {
                        for(j=0;j<aa[k];j++) //j recruited bee
                        {
                                for(d=0;d<dim;d++)//d dimension
                                {
                                        if     ((candidx[d][k]-nghx[d])<start_x[d])     //
boundry check (left)
                                                                                bNghPos[d]
[j]=randfunc(start_x[d],candidx[d][k]+nghx[d]);
                                          else   if   ((candidx[d][k]+nghx[d])>end_x[d])//
boundry check (right)
                                                bNghPos[d][j]=randfunc(candidx[d][k]-
nghx[d],end_x[d]);
                                     else
                                                bNghPos[d][j]=randfunc(candidx[d][k]-
nghx[d] , candidx[d][k]+nghx[d]);
                                     }
                                }// end of recruitment

                         for(j=0;j<aa[k];j++)  // evaluate fitness for recruited bees
                                 bNghFit[j]=func(bNghPos,j);

                         for(j=0;j<aa[k];j++)                    // choosing the rep bee
                                 if(bNghFit[j]>= temp1 )
                                        {
                                                temp1=bNghFit[j];
                                                temp2=j;
                                                }  // end of choosing the rep bee

                        for(d=0;d<dim;d++)
                                bPos[d][counter1]=bNghPos[d][temp2];

                        counter1++; // next member of the new list

                        temp1=-INT_MAX;             //

                }       // end of  Neighbourhood Search

                 for(k=0;k<ranSearchBees;k++)       //start of rand search for rest of
bees
                         {
                                for(d=0;d<dim;d++)
                                        bPos[d]
[counter1]=randfunc(start_x[d],end_x[d]);
                                counter1++;
                                }
```

```
                for(j=0;j<n;j++)                    // evalute the fitness of the new list
                        fit[j]=func(bPos,j);
                gen_fit[iter][runs]=-fit[0];
                }          //end iter = imax

 fprintf(Array,"%d,  %f,  %f,  %f \n",runs+1,fit[0],bPos[0][0],bPos[1][0]);
} // end runs

        Result.close();

        cout<<"\n finished.....\n";

}
```

Subroutines:

```
// Parameters used in main.cpp

#define dim 2       // Dimension of function
// Variables used
#define pop  500                    // max num of population
        int imax=100;
        int n= 45;  // Number of Scout Bees
        int m= 3;   // Number of selected Locations
        int n1 =2;  // Number of Bees around each selected locations ( except the elite
location )
        int n2 =7;  // Number of Bees around each ellit locations
        int e =1;   // Elite point/s
        double ngh=.06;  // Neighbourhood search domain

        double q0=0.90; // Balancing b/w Exploration ( Probabalistically choosing m
sites)
                    // and Exploitation ( choosing best m sites )



// Function used in main.cpp
 #include <iostream.h>

double start_x[]={-65.536,-65.536};
double end_x[]={65.536,65.536};
double ans = -0.9982;
double func(double x[][pop], int i )  // Definition of Fitness Function

{
  double y;

double  a[25][2]={{-32,-32},{-16,-32},{0,-32},{16,-32},{32,-32},{-32,-16},{-16,-16},
{0,-16},{16,-16},{32,-16},{-32,0},{-16,0},{0,0},{16,0},{32,0},{-32,16},{-16,16},
{0,16},{16,16},{32,16},{-32,32},{-16,32},{0,32},{16,32},{32,32}};
```

```
        y =0.002;

        for(int j=0;j<25;j++)
        {
                y=y+1/((j+1)+pow(x[0][i]-a[j][0],6)+pow(x[1][i]-a[j][1],6));
        }
 y=-1/y;
 return y;
}

double randfunc( double xs, double xe )  // Definition of Randon number Generator Function
{
 double randnum;
 randnum=rand() ;
 randnum=xs+randnum*(xe-xs) / RAND_MAX;
 return randnum;
}
double myrandom()
{
        double r;
        int M,x;
        M = 10000;
        x= M-2;
        r = (1.0+(rand()%x))/M;
        return r;
}

void funcSort(double inP1[],double oP1[],double inP2[][pop],double oP2[][pop],int size)
{
                int temp2;
                double temp1=-INT_MAX;
                        for( int j=0;j<size;j++)          //sort
                        {
                                for(int k=0;k<size;k++)
                                        if(inP1[k]> temp1 )
                                        {
                                                temp1=inP1[k];
                                                temp2=k;
                                        }

                                oP1[j]=temp1;

                                for(int d=0;d<dim;d++)
                                  oP2[d][j]=inP2[d][temp2];

                                temp1=-INT_MAX;
```

```
                    inP1[temp2]=-INT_MAX;

                }       //end sort

}
```

References

[1] Bilchev G. and Parmee IC. The Ant Colony Metaphor for Searching Continuous Design Spaces, **Proceedings of the AISB Workshop on Evolutionary computation, University of Sheffield, UK, April 3 - 4, 1995.**

[2] Mathur M, Karale SB., Priye S., Jayaraman VK. and Kulkarni BD. Ant Colony Approach to Continuous Function Optimization, Industrial & Engineering Chemistry Research 2000, 39, 3814 – 3822

[3] Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, MA, 1997.

[4] Dorigo M. and Stützle T. Ant Colony Optimization, Cambridge, Mass. ; London : MIT Press, 2004.

[5] Kennedy J. and Ebernhart R. Swarm Intelligence, Morgan Kaufmann Publishers, San Francisco, 2001

[6] Tovey CA. The honey bee algorithm, a biologically inspired approach to internet server optimization. Engineering Enterprise, Spring 2004, pp.13-15.

[7] Yang, X. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms, IWINAC 2005, Lecture Notes in Computer Science, volume 3562, pp. 317-323.

[8] Wedde HF., Farooq M. and Zhang Y. BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior. ANTS 2004, LNCS 3172, pp.83–94.

[9] Wedde HF., Farooq M., Pannenbaecker T., Vogel B., Mueller C., Meth J., and Jeruschkat R. BeeAdHoc: An Energy Efficient Routing Algorithm for Mobile AdHoc Networks Inspired by Bee Behaviour. GECCO'05, June 25–29, USA. 2005.

[10] Bonabeau E., Dorigo M. and Theraulaz G. Swarm Intelligence from Natural to Artificial Systems. Oxford University Press, New York. 1999.

[11] Camazine S., Deneubourg J., Franks NR., Sneyd J., Theraulaz G. and Bonabeau E. Self-Organization in Biological Systems. Princeton University Press, Princeton, 2001.

[12] Frisch KV. Bees: their vision, chemical senses, and language. Cornell Paperbacks publishing, 1967.

[13] Seeley TD. The Wisdom of the Hive - The Social Physiology of Honey Bee Colonies. Harvard University Press, Cambridge, Massachusetts, 1995.