Available online at www.worldscientificnews.com

World Scientific News

WSN 12 (2015) 41-56 EISSN 2392-2192

Solving Banana (Rosenbrock) Function Based on Fitness Function

Lubna Zaghlul Bashir[1,a], Rajaa Salih Mohammed Hasan[2,b]

[1]Building and Construction Department, Technology University, Baghdad, Iraq

[2]Al Mansur Institute of Medical Technology, Middle Technical University, Iraq

[a,b]E-mail address: lubna_zaghlul@yahoo.com , raja.salih@ymail.com

ABSTRACT

When solving real optimization problems numerically, the solution process typically involves the phases of modelling, simulation and optimization. A simulated model of a real life problem is often complex, and the objective function to be minimized may be non convex and have several local minima. Then, global optimization methods are needed to prevent the stagnation to a local minimum. Therefore, in the recent years, there has been a great deal of interest in developing methods for solving global optimization problems [1, 2, 3] and references therein). Genetic algorithms [4, 5, 6] are meta heuristics used for solving problems with both discrete and continuous variables. The population is the main element of genetic algorithms, and the genetic operations like crossover and mutation are just instruments for manipulating the population so that it evolves towards the final population including a "close to optimal" solution. The requirements set on the population also change during the execution of the algorithm [7]. In this work we test the function known as the "banana function" because of its shape; $f(x) = 100. (x2 - x1^2)^2 + (1- x1)^2$. In this problem, there are two design variables with lower and upper limits of [-5 , 5].we use genetic algorithm for solving this problem, Basic philosophy of genetic algorithm and its flowchart are described. Step by step numerical computation of genetic algorithm for solving the banana function will be briefly explained. The results shows that The Rosenbrock function has a known global minimum at [1 , 1] with an optimal function value of zero.

***Keywords*:** Banana Function; Genetic Algorithm; Optimization; Fitness; Evolutionary; Generations

1. INTRODUCTION

Genetic algorithms are stochastic search techniques that guide a population of solutions towards an optimum using the principles of evolution and natural genetics. In recent years, genetic algorithms have become a popular optimization tool for many areas of research, including the field of system control, control design, science and engineering. Significant research exists concerning genetic algorithms for control design and off-line controller analysis.

Genetic algorithms are inspired by the evolution of populations. In a particular environment, individuals which better fit the environment will be able to survive and hand down their chromosomes to their descendants, while less fit individuals will become extinct. The aim of genetic algorithms is to use simple representations to encode complex structures and simple operations to improve these structures. Genetic algorithms therefore are characterized by their representation and operators. In the original genetic algorithm an individual chromosome is represented by a binary string. The bits of each string are called genes and their varying values alleles. A group of individual chromosomes are called a population. Basic genetic operators include reproduction, crossover and mutation. Genetic algorithms are especially capable of handling problems in which the objective function is discontinuous or non differentiable, non convex, multimodal or noisy. Since the algorithms operate on a population instead of a single point in the search space, they climb many peaks in parallel and therefore reduce the probability of finding local minima [8].

2. GENETIC ALGORITHM PERFORMANCE

- GAs can provide solutions for highly complex search spaces
- GAs perform well approximating solutions to all types of problems because they do not make any assumption about the underlying fitness landscape(the shape of the fitness function, or objective function)
- However, GAs can be outperformed by more field-specific algorithms [9].

3. BASIC PHILOSOPHY

Genetic algorithm developed by Goldberg was inspired by Darwin's theory of evolution which states that the survival of an organism is affected by rule "the strongest species that survives". Darwin also stated that the survival of an organism can be maintained through the process of reproduction, crossover and mutation. Darwin's concept of evolution is then adapted to computational algorithm to find solution to a problem called objective function in natural fashion.

A solution generated by genetic algorithm is called a chromosome, while collection of chromosome is referred as a population. A chromosome is composed from genes and its value can be either numerical, binary, symbols or characters depending on the problem want to be solved. These chromosomes will undergo a process called fitness function to measure the suitability of solution generated by GA with problem.

Some chromosomes in population will mate through process called crossover thus producing new chromosomes named offspring which its genes composition are the combination of their parent. In a generation, a few chromosomes will also mutation in their gene. The number of chromosomes which will undergo crossover and mutation is controlled by crossover rate and mutation rate value. Chromosome in the population that will maintain for the next generation will be selected based on Darwinian evolution rule, the chromosome which has higher fitness value will have greater probability of being selected again in the next generation. After several generations, the chromosome value will converges to a certain value which is the best solution for the problem [10]. Table 1 illustrate genetic algorithm terms [11].

Table 1. Genetic Algorithm Terms

Biological	GA Terms
Chromosome	String
Gene	Feature, character
Genomes	Guesses, solutions, collection of genes

4. FITNESS FUNCTION

- Fitness is computed for each individual
- To help maintain diversity or differentiate between similar individuals, raw objective scores are sometimes scaled to produce the final fitness scores
- Rank - no scaling
- Linear scaling (fitness proportionate) – normalizes based on min and max fitness in population
- Sigma truncation scaling - normalizes using population mean and std. dev., truncating low-fitness individuals
- Sharing (similarity scaling) - reduces fitness for individuals that are similar to other individuals in the population [12].

5. GENETIC ALGORITHM OPERATORS

The primary work of the Simple Genetic Algorithm (SGA) is performed in three routines:

1. Select.
2. Crossover.
3. Mutation.

Their action is coordinate by a procedure called generation that generates a new population at each successive generation [13].

5. 1. Selection

The purpose of selection is, of course, to emphasize the fitter individuals in the population in hopes that their offspring will in turn have even higher fitness.
Methods of selection

- Roulette wheel selection.
- Sigma scaling techniques.
- Tournament selection.
- Ranking methods.
- Elitism.
- Boltzmann selection.
- Steady state selection.

Holland's original GA used fitness- proportionate selection, in which the expected value of an individual (i.e., the expected number of times an individual will be selected to reproduce) is that individuals fitness is divided by the average fitness of population. The most common method to implement this is 'roulette wheel" sampling [13,14].

Roulette Wheel Selection

Each individual is assigned a slice of a circular "roulette wheel" the size of the slice being proportional to the individual's fitness. The wheel is spun N times, where N is the number of individuals in the population. On each spin, the individual under the wheel's marker is selected to be in the pool of parents for the next generation [9]. Figure 1, illustrate Roulette Wheel Selection [15].

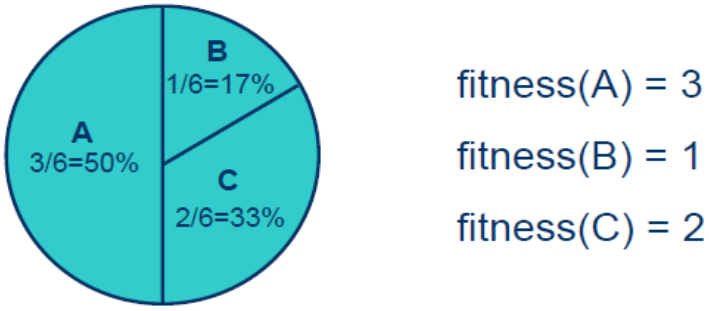

Figure 1. Roulette Wheel Selection

Other section methods

- *Elitist selection*:

Chose only the most fit members of each generation.

- *Cut off selection*:

Select only those that are above a certain cut off for the target function [14].

5. 2. Crossover

In contrast is applied with high probability. It is a randomized yet structured operator that allows information exchange between points. Simple crossover is implemented by choosing a random point in the selected pair of strings and exchanging the sub strings defined by that point. Figure. 2 shows how Crossover mixes information from two parent strings, producing offspring made up of parts from both parents. This operator which does no table lookups or backtracking, is very efficient because of its simplicity [5].

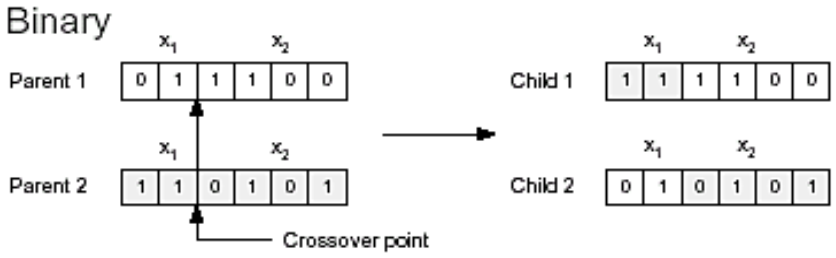

Figure 2. Crossover Operator.

Several possible crossover strategies

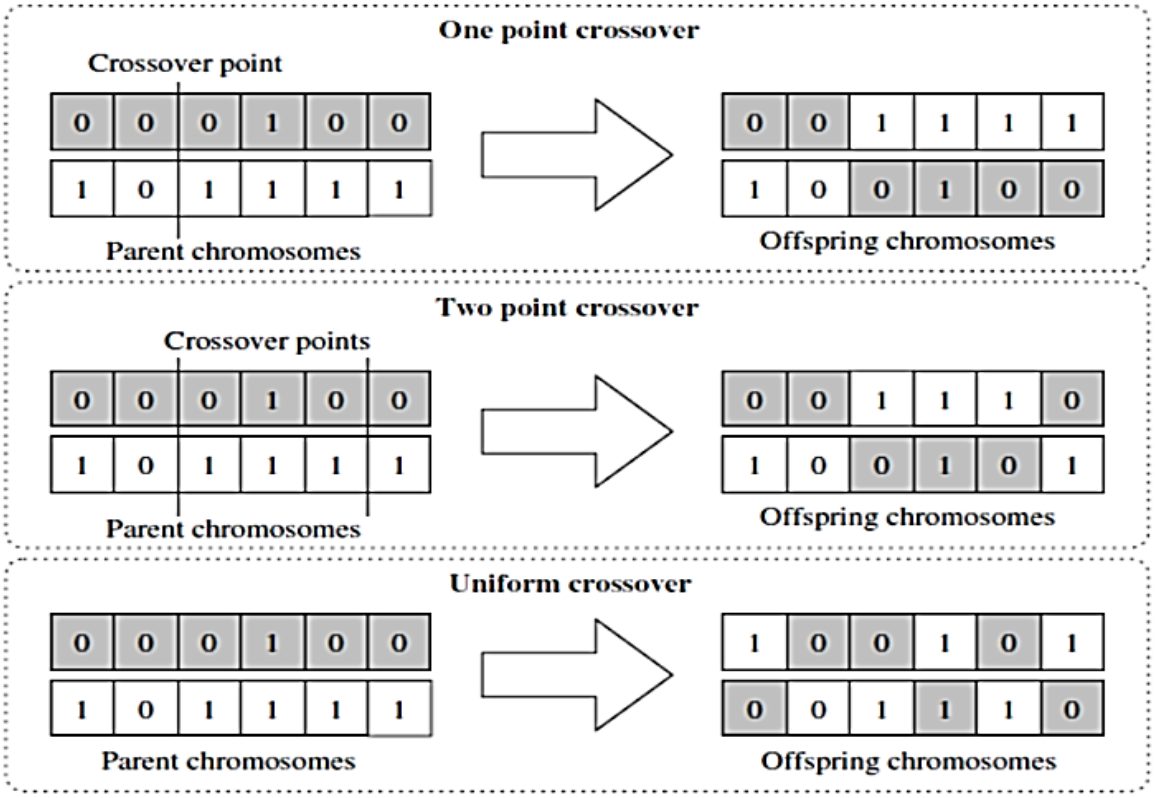

Figure 3. One-point, two-point, and uniform crossover methods.

1. Randomly select a single point for a crossover.
2. Multi point crossover. Avoids cases where genes at the beginning and end of a chromosome are always split
3. Uniform crossover. A random subset is chosen The subset is taken from parent 1 and the other bits from parent2.Figure.3.illistrate crossover methods [14,16].

5. 3. Mutation

As in natural systems, is a very low probability operator and just flips a specific bit. Where the bits at one or more randomly selected positions of the chromosomes are altered. The mutation process helps to overcome trapping at local maxima. Figure. 4 shows the effect of the one-bit mutation operator in the binary case [5].

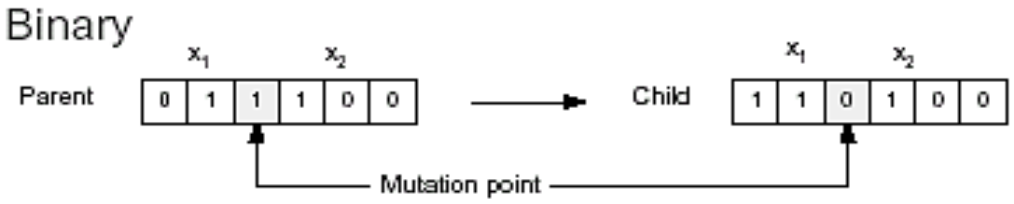

Figure 4. Mutation operator

6. GENETIC ALGORITHM CODE

Given a clearly defined problem to be solved and a bit string representation for candidate solutions, a simple GA works as follows:

1. Start with a randomly generated population of n l−bit chromosomes (candidate solutions to a problem).

2. Calculate the fitness $f(x)$ of each chromosome x in the population.

3. Repeat the following steps until n offspring have been created:

a. Select a pair of parent chromosomes from the current population, the probability of selection being an increasing function of fitness. Selection is done "with replacement," meaning that the same chromosome can be selected more than once to become a parent.

b. With probability pc (the "crossover probability" or "crossover rate"), cross over the pair at a randomly chosen point (chosen with uniform probability) to form two offspring. If no crossover takes place, form two offspring that are exact copies of their respective parents. (Note that here the crossover rate is defined to be the probability that two parents will cross over in a single point. There are also "multi−point crossover" versions of the GA in which the crossover rate for a pair of parents is the number of points at which a crossover takes place.)

c. Mutate the two offspring at each locus with probability pm (the mutation probability or mutation rate), and place the resulting chromosomes in the new population. If n is odd, one new population member can be discarded at random.

4. Replace the current population with the new population.

5. Go to step 2.

Figure 5 illustrate genetic algorithm code and Figure 6 illistrates genetic algorithm diagram [17,18,19]

.

```
begin
        t=0
        initialize Pt
        evaluate Pt
        while(termination condition not satisfied) do
        begin
                t=t+1
                select Pt from Pt-1
                mutate Pt
                evaluate Pt
        end
end
```

Figure 5. Genetic Algorithm Code

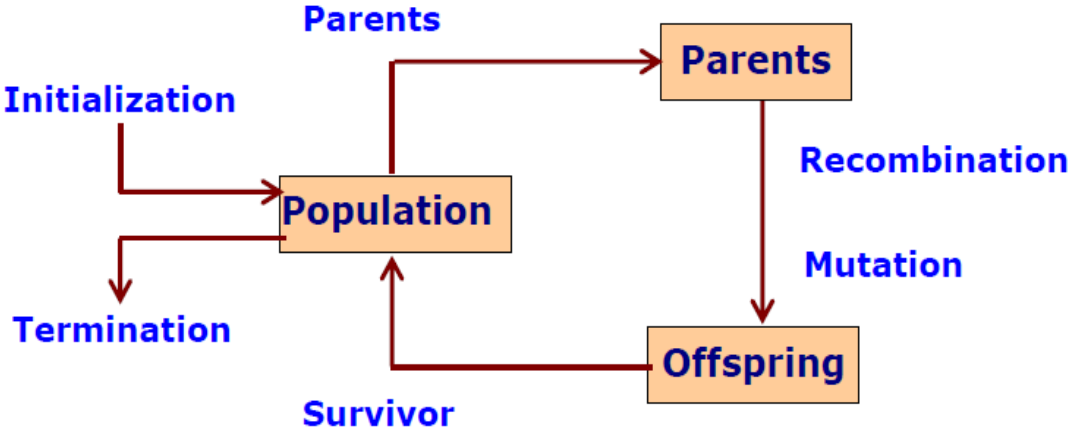

Figure 6. Genetic algorithm diagram.

7. THE BANANA (ROSENBROCK) FUNCTION

This function is known as the "banana function" because of its shape; it is described mathematically in Equation 1. In this problem, there are two design variables with lower and upper limits of [-5 , 5].

$$\text{MINIMIZE } F(X) = 100.\ (X2-X1^2)^2 + (1-X1)^2 \qquad (1)$$

7. 1. Initialization

We define the number of chromosomes in population are 6, then generate random value of gene X1,X2 for 6 chromosomes

Chromosome[1] = [X1,X2] = [2;5]
Chromosome[2] = [X1,X2] = [2;1]
Chromosome[3] = [X1,X2] = [3;4]
Chromosome[4] = [X1,X2] = [2;0]
Chromosome[5] = [X1,X2] = [1;4]
Chromosome[6] = [X1,X2] = [5;1]

7. 2. Evaluation

To compute the objective function value for each chromosome produced in initialization step:

F_obj[1] = 100*(5-2^2)^2 + (1-2)^2 = 101
F_obj[2] = 100*(1-2^2)^2 + (1-2)^2 = 901
F_obj[3] = 100*(4-3^2)^2 + (1-3)^2 = 2504
F_obj[4] = 100*(0-2^2)^2 + (1-2)^2 = 1601
F_obj[5] = 100*(4-1^2)^2 + (1-1)^2 = 900
F_obj[6] = 100*(1-5^2)^2 + (1-5)^2 = 57616

7. 3. Selection

The fittest chromosomes have higher probability to be selected for the next generation. To compute fitness probability we must compute the fitness of each chromosome.

Fitness[1] = 1 / F_obj[1] = 1 / 101 = 0.0099
Fitness[2] = 1 / F_obj[2] = 1 / 901 = 0.0011
Fitness[3] = 1 / F_obj[3] = 1 / 2504 = 0.0003
Fitness[4] = 1 / F_obj[4] = 1 / 1601 = 0.0006
Fitness[5] = 1 / F_obj[5] = 1 / 900 = 0.0011
Fitness[6] = 1 / F_obj[6] = 1 / 57616 = 0.00001
Total = 0.01316

The probability for each chromosomes is formulated by: P[i] = Fitness[i] / Total

P[1] = 0.0099 / 0.01316 = 0.75216
P[2] = 0.0011 / 0.01316 = 0.08431

P[3] = 0.0003 / 0.01316 = 0.03033
P[4] = 0.0006 / 0.01316 = 0.04745
P[5] = 0.0011 / 0.01316 = 0.08440
P[6] = 0.00001 / 0.01316 = 0.00131

From the probabilities above we can see that Chromosome 1 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes. For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

C[1] = 0.75216
C[2] = 0.75216 + 0.08431 = 0.83648
C[3] = 0.75216 + 0.08431 + 0.03033 = 0.86682
C[4] = 0.75216 + 0.08431 + 0.03033 + 0.04745 = 0.91427
C[5] = 0.75216 + 0.08431 + 0.03033 + 0.04745 + 0.08440 = 0.99868
C[6] = 0.75216 + 0.08431 + 0.03033 + 0.04745 + 0.08440 + 0.00131 = 1

7. 4. Crossover

In this example, we use one-cut point, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome. Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using crossover_rate (ρc) parameters. Pseudo-code for the crossover process is as follows:

```
begin
k← 0;
while (k<population) do
R[k] ← random(0-1);
if (R[k] < ρc) then
select Chromosome[k] as parent;
end;
k = k + 1;
end;
```

Chromosome k will be selected as a parent if R [k] <ρc. Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25. The process is as follows: First we generate a random number R as the number of population.

R[1] = 0.202
R[2] = 0.360
R[3] = 0.871
R[4] = 0.117
R[5] = 0.260
R[6] = 0.451

For random number R above, parents are Chromosome [1], Chromosome [2] and Chromosome [5] will be selected for crossover.
Chromosome [1] >< Chromosome [2]

Chromosome[1] >< Chromosome[5]
Chromosome[5] >< Chromosome[2]

Chromosome[1] >< Chromosome[2] = [2,5] >< [2,1]
Offspring1 = [2,2], Offspring2 = [5,1]
Chromosome[1] >< Chromosome[5] = [2,5] >< [1,4]
Offspring3 = [2,1], Offspring4 = [5,4]
Chromosome[5] >< Chromosome[2] = [1,4] >< [2,1]
Offspring5 = [1,2], Offspring6 = [4,1]

Thus Chromosome population after experiencing a crossover process:

Offspring [1] = [2,2]
Offspring [2] = [5,1]
Offspring [3] = [2,1]
Offspring [4] = [5,4]
Offspring [5] = [1,2]
Offspring [6] = [4,1]

7. 5. Mutation

Number of chromosomes that have mutations in a population is determined by the mutation rate parameter. Mutation process is done by replacing the gen at random position with a new value. The process is as follows. First we must calculate the total length of gen in the population. In this case the total length of gen is total_gen = number_of_gen_in_Chromosome * number of population
= 2 * 6
= 12

Mutation process is done by generating a random integer between 1 and total_gen (1 to 12). If generated random number is smaller than mutation_rate (ρm) variable then marked the position of gen in chromosomes. Suppose we define ρm 10%, it is expected that 10% (0.1) of total_gen in the population that will be mutated:
number of mutations = 0.1 * 12
= 1.2
≈ 1

Suppose generation of random number yield 7 and 10 then the chromosome which have mutation are Chromosome number 4 gen number 5 and Chromosome 5 gen number 2. The value of mutated gens at mutation point is replaced by random number between -5,5. Suppose generated random number are 0 and 1 then Chromosome composition after mutation are:

Offspring [1] = [2,2]
Offspring [2] = [5,1]
Offspring [3] = [2,1]
Offspring [4] = [0,4]
Offspring [5] = [1,1]
Offspring [6] = [4,1]

Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:
offspring [1] = [2,2]

F_obj [1] = 100*(2-2^2)^2+(1-2)^2 = 401
offspring [2] = [5,1]
F_obj[2] = 100*(1-5^2)^2+(1-5)^2 = 57616
offspring [3] = [2,1]
F_obj[3] = 100*(1-2^2)^2+(1-2)^2 = 901
offspring [4] = [0,4]
F_obj[4] = 100*(4-0^2)^2+(1-0)^2 = 1601
offspring [5] = [1,1]
F_obj[5] = 100*(1-1^2)^2+(1-1)^2 = 0
offspring [6] = [4,1]
F_obj[6] = 100*(1-4^2)^2+(1-4)^2 = 22509

From the evaluation of new offsprings we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous Chromosome generation. chromosome[6] must change by offspring[5], chromosome[3] must change by offspring[1], chromosome[4] must change by offspring[3],

New Chromosomes for next iteration are:

Chromosome[1] = [2,5]
Chromosome[2] = [2,1]
Chromosome[3] = [2,2]
Chromosome[4] = [2,1]
Chromosome[5] = [1,4]
Chromosome[6] = [1,1]

Chromosome [1] = [2,5]

F_obj [1] = 100*(5-2^2)^2+(1-2)^2 = 101
Chromosome [2] = [2,1]
F_obj[2] = 100*(1-2^2)^2+(1-2)^2 = 901
Chromosome [3] = [2,2]
F_obj[3] = 100*(2-2^2)^2+(1-2)^2 = 401
Chromosome [4] = [2,1]
F_obj[4] = 100*(2-1^2)^2+(1-2)^2 = 901
Chromosome [5] = [1,4]
F_obj[5] = 100*(4-1^2)^2+(1-1)^2 = 900
Chromosome [6] = [1,1]
F_obj[6] = 100*(1-1^2)^2+(1-1)^2 = 0

These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration. This process will be repeated until a

predetermined number of generations. For this function, after running 50 generations, best chromosome is obtained:

Chromosome = [1,1]

This means that:

X1 = 1, X2 = 1,

If we use the number in the banana problem

$F(X) = 100.\ (X2-X1^2)^2 + (1-X1)^2$

8. EXPERIMENTAL RESULT

Executing the simple genetic algorithm code (SGA) written in Pascal programming language, the program call three routines, selection, crossover, mutation as illustrated in Figures 7,8,9 respectively.

```
Function Select
        Begin
            Partsum:=0.0;j:=0;
            Rand:=random*sumfitness;
            Repeat
            J:=j+1;
            Partsum:=partsum+pop[j].fitness;
           Until(partsum >= rand) or (j:=popsize);
           Select j;
        End;
```

Figure 7. Function Select

```
Procedure crossover
Begin
If flip (pcross) then begin
    Jcross:=rnd(1,lchrom-1);
    Ncross:=ncross+1;
    End else
    Jcross:=lchrom;
    For j:=1 to jcross do begin
        Child1[j]:=mutation(parent1[j],pmutation,nmutation);
        Child2[j]:=mutation(parent2[j],pmutation,nmutation);
    end;
    If jcross<> lchrom then
   For j:=jcross+1 to lcross do begin
        Child1[j]:=mutation(parent2[j],pmutation,nmutation);
        Child2[j]:=mutation(parent1[j],pmutation,nmutation);
    end;
end;
```

Figure 8. Function Crossover

```
Function mutation
Begin
        Mutate:= flip(pmutation);
        If mutate then begin
                Nmutation:=nmutation+1;
                Mutation:=notallelval
         End else
                Mutation:=allelval;
End;
```

Figure 9. Function Mutation

The primary data structure is the population of 36 string and the following parameters :

SGA Parameters

Population size = 36
Chromosome length = 2
Maximum of generation = 20
Crossover probability = 0.25
Mutation probability = 0.01

Population

Table 2. Illustrate initial population

Table 2. Initial Population

No.	X1	X2
1.	0	0
2.	1	0
3.	2	0
4.	3	0
5.	4	0
6.	5	0
7.	0	1
8.	1	1
9.	2	1
10.	3	1
11.	4	1

12.	5	1
13.	0	2
14.	1	2
15.	2	2
16.	3	2
17.	4	2
18.	5	2
19.	0	3
20.	1	3
21.	2	3
22.	3	3
23.	4	3
24.	5	3
25.	0	4
26.	1	4
27.	2	4
28.	3	4
29.	4	4
30.	5	4
31.	0	5
32.	1	5
33.	2	5
34.	3	5
35.	4	5
36.	5	5

The Rosenbrock function has a known global minimum at [1 , 1] with an optimal function value of zero.

9. CONCLUSIONS

- Fitness scaling has been suggested for maintaining more careful control over the allocation of trails to the best string.
- Many practical optimization problems require the specification of a control function or functions over a continuum.

- GA must have fitness functions that reflect information about both the quality and feasibility of solutions.
- Survival of the fittest, the most fit of a particular generation (the nearest to a correct answer) are used as parents for the next generation.
- The most fit of this new generation are parents for the next, and so on. This eradicates worse solutions and eliminates bad family lines.
- The GA does find near optimal results quickly after searching a small portion of the search space.
- Results shows that the genetic algorithm have the ability to find optimal solution for solving banana (rosenbrock) function.

References

[1] Floudas C.A., Pardalos, " RecentAdvances inGlobalOptimization". PrincetonUniversity Press, Princeton, NJ (1992).

[2] Horst R., Pardalos, Handbook of "Global Optimization". Kluwer Academic Publishers, Dordrecht (1995).

[3] Pardalos P.M., Romeijn H.E. Handbook of" Global Optimization, Vol. 2. Kluwer Academic Publishers, Boston (2002).

[4] Glover F., Kochenberger G. A., Handbook of "Metaheuristics". KluwerAcademic Publishers, Boston (2003).

[5] Goldberg D. E., Genetic Algorithms in Search, Optimization, and Machine Learning. Addison- Wesley, New York (1989).

[6] Michalewicz Z., "Genetic Algorithms + Data Structures = Evolution Programs". Springer-Verlag, Berlin (1994).

[7] Heikki Maaranen, Kaisa Miettinen, Antti Penttinen, "On initial populations of a genetic algorithm for continuous optimization problems', (2007).

[8] Ms. Dharmistha, D. Vishwakarma," Genetic Algorithm based Weights Optimization of Artificial Neural Network", *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering,* Vol. 1, Issue 3, August (2012).

[9] Olesya Peshko, "Global Optimization Genetic Algorithms", (2007).

[10] Denny Hermawanto," Genetic Algorithm for Solving Simple Mathematical Equality Problem", Indonesian Institute of Sciences (LIPI), INDONESIA, (2013).

[11] Damian Schofield, Rong Qu,"genetic algorithms", (2012).

[12] Larry Yaeger, "Intro to Genetic Algorithms" Artificial Life as an approach to Artificial Intelligence" Professor of Informatics, Indiana University, (2008).

[13] Zhou Qing Qing and Purvis. Martin, "A Market-Based Rule Learning System" Guang Dong Data Communication Bureau China Telecom 1 Dongyuanheng Rd., Yuexiunan,

Guangzhou 510110, China, Department of Information Science, University of Otago, PO Box 56, Dunedin, New Zealand and/or improving the comprehensibility of the rules, 2004.

[14] Abo Rajy ,"Genetic algorithms", (2013).

[15] Bull Larry, "Learning Classifier Systems: A Brief Introduction", Faculty of Computing, Engineering & Mathematical Sciences University of the West of England, Bristol BS16 1QY, U.K. Larry, (2004).

[16] Kumara Sastry, David Goldberg, Graham Kendall, "GENETIC ALGORITHMS", University of Illinois, USA, University of Nottingham, UK, 2011.

[17] Damian Schofield, Rong Qu,"Genetic algorithms", 2012.

[18] Mitchell Melanie," An Introduction to Genetic Algorithms", A Bradford Book The MIT Press Cambridge, Massachusetts, London, England, Fifth printing, 1999.

[19] Saif Hasan, Sagar Chordia, Rahul Varshneya,"Genetic algorithm", February 6, 2012.

[20] R. C. Chakraborty "Fandemantaels of genetic algorithms": AI course lecture 39-40, notes, slides, 2010.

(Received 29 May 2015; accepted 11 June 2015)