Available online at www.worldscientificnews.com

World Scientific News

WSN 19 (2015) 148-167 EISSN 2392-2192

Solve Simple Linear Equation using Evolutionary Algorithm

Lubna Zaghlul Bashir

Building and Construction Department, Technology University, Baghdad, Iraq

E-mail address: lubna_zaghlul@yahoo.com

ABSTRACT

Modern programs present a large number of optimization options covering the many alternatives to achieving high performance for different kinds of applications and workloads. Selecting the optimal set of optimization options for a given application and workload becomes a real issue since optimization options do not necessarily improve performance when combined with other options. In this work we use genetic algorithms to solve linear equation problem. Suppose there is equality a + 2 b + 3 c = 10, genetic algorithm will be used to find the value of a, b and c that satisfy the above equation. 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. Results shows that many practical optimization problems require the specification of a control function, and The GA does find near optimal results quickly after searching a small portion of the search space.

Keywords: Genetic Algorithm; Population; Evolutionary Algorithm; Objective Function

1. INTRODUCTION

In modern compilers and optimizers, the set of possible optimizations is usually very large. In general, for a given application and workload, optimization options do not accrue toward ultimate performance. To avoid selection complexity, users tend to use standard combination options, in general, however, these standard options are not the optimal set for a specific application executing its representative workload [1,2]. 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 [3-5].

2. OPTIMIZATION TECHNIQUES

2. 1. Analytical

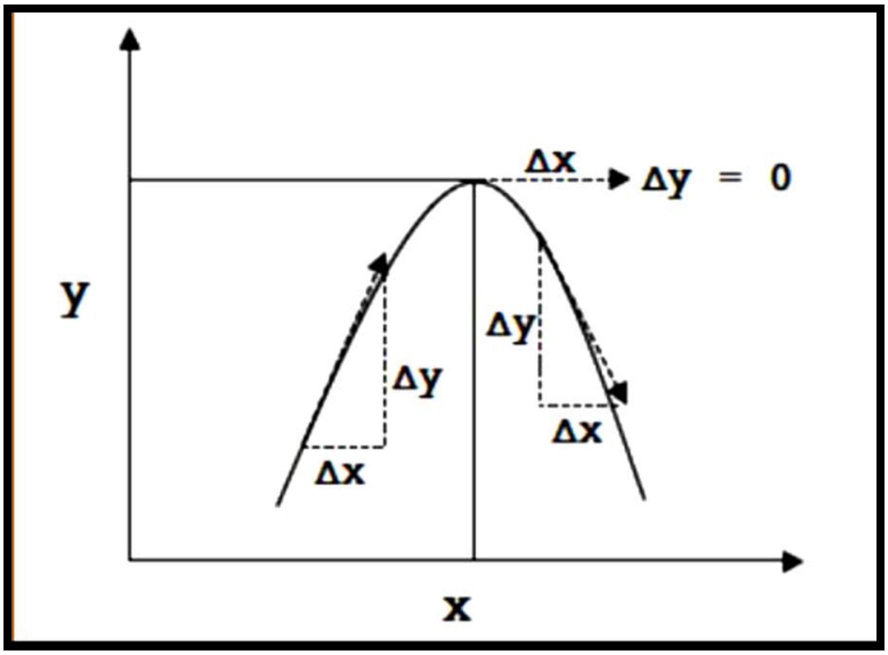

Figure 1. Analytical Technique.

Given y = f(x), take the derivative of w.r.t. x, set the result to zero, solve for x, Works perfectly, but only for simple, analytical Functions.as illustrated in Figure 1.

2. 2. Gradient-based, or hill-climbing

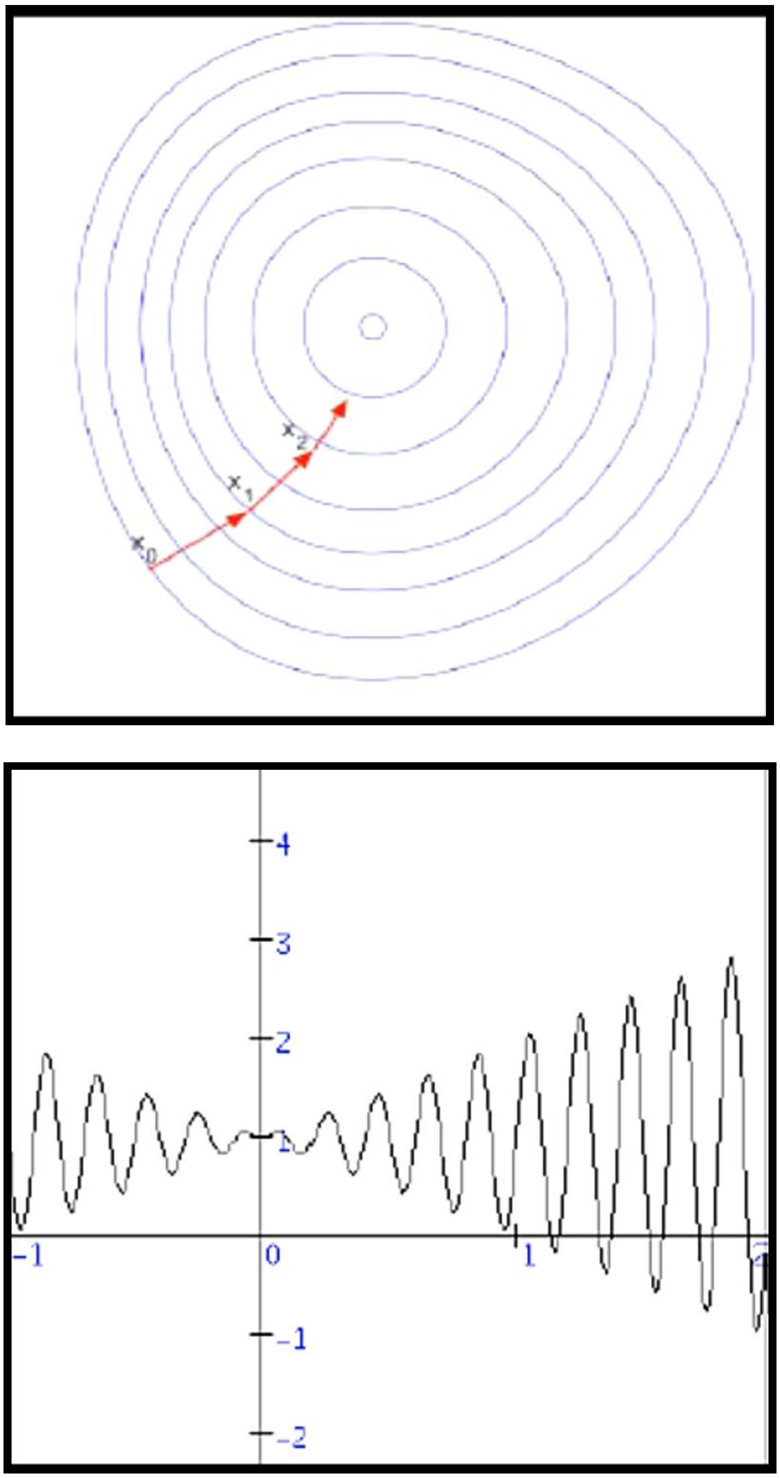

Figure 2. Hill- Climbing Technique.

Given y = f(x)

- pick a point x0
- compute the gradient $\nabla f(x0)$
- step along the gradient to obtain $x1 = x0 + \alpha \nabla f(x0)$.
- repeat until extreme is obtained

Requires existence of derivatives, and easily gets stuck on local extreme. Figure 2 illustrate hill-climping technique.

2. 3. Enumerative

Test every point in the space in order.

2. 4. Random

Test every point in the space randomly.

2. 5. Genetic Algorithm (Evolutionary Computation)

• Does not require derivatives, just an evaluation function (a fitness function).
• Samples the space widely, like an enumerative or random algorithm, but more efficiently.
• Can search multiple peaks in parallel, so is less hampered by local extreme than gradient-based methods.
• Crossover allows the combination of useful building blocks, or schemata (mutation avoids evolutionary Dead-ends).
• Robust [6,7].

3. GENETIC ALGORITHMS FOR OPTIMIZATION

Optimization is a process that finds a best, or optimal solutions for a problem. The optimization problems are catered around three factors:

1. **An objective function:** which is to be minimize or maximize.
2. **A set of unknowns or variables:** that effect the objective function.
3. **A set of constraints:** that allow the unknowns to take on certain values.

An optimization problem is defined as finding values of the variables that minimize or maximize the objective function while satisfying the constraints but exclude others [8].

The Genetic Algorithms are direct, stochastic method for optimization. Since they use populations with allowed solutions (individuals), they count in the group of parallel algorithms. Due to the stochastic was of searching, in most cases, it is necessary to set limits at least for the values of the optimized parameters [4].

A genetic algorithm developed by J.H. Holland, 1975 [3]. Genetic algorithms are stochastic search techniques that guide a population of solutions towards an optimum using the principles of evolution and natural genetics [2].

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 [2].

4. OPTIMIZATION PROBLEMS

Genetic algorithms evaluate the target function to be optimized at some randomly selected points of the definition domain. Taking this information into account, a new set of points (a new population) is generated. Gradually the points in the population approach local maxima and minima of the function.

Genetic algorithms can be used when no information is available about the gradient of the function at the evaluated points. The function itself does not need to be continuous or differentiable. Genetic algorithms can still achieve good results even in cases in which the function has several local minima or maxima. These properties of genetic algorithms have their price: unlike traditional random search, the function is not examined at a single place, constructing a possible path to the local maximum or minimum, but many different places are considered simultaneously. The function must be calculated for all elements of the population. The creation of new populations also requires additional calculations. In this way the optimum of the function is sought in several directions simultaneously and many paths to the optimum are processed in parallel. The calculations required for this feat are obviously much more extensive than for a simple random search [3,9].

5. 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 [10,11].

6. SCHEME OF THE EVOLUTIONARY ALGORITHMS

The EA holds a population of individuals (chromosomes), which evolve means of selection and other operators like crossover and mutation. Every individual in the population gets an evaluation of its adaptation (fitness) to the environment. In the terms of optimization this means, that the function that is maximized or minimized is evaluated for every individual. The selection chooses the best gene combinations (individuals), which through crossover and mutation should drive to better solutions in the next population [4].

7. GENETIC ALGORITHM TERMS

a. Gene - a single encoding of part of the solution space, i.e. either single bits or short blocks of adjacent bits that encode an element of the candidate solution. Figure 3 shows gene representation [10].

Figure 3. Gene representation.

b. Chromosome - a string of genes that represents a solution. Figure 4 shows chromosome representation [10].

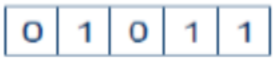

Figure 4. Chromosome representation.

Chromosomes can be:

- Bit strings (0110, 0011, 1101, …)
- Real numbers (33.2, -12.11, 5.32, …)
- Permutations of element (1234, 3241, 4312, …)
- Lists of rules (R1, R2, R3, …Rn…)
- Program elements (genetic programming)
- Any other data structure

c. Population - the number of chromosomes available to test. Figure 5 shows population representation [10,12].

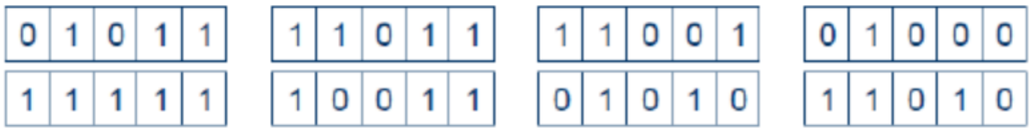

Figure 5. Population representation.

d. Fitness

- 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 [6,12].

e. Selection: The selection scheme determines how individuals are chosen for mating, based on their fitness scores.

Methods of selection

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

- Boltzmann selection.
- Steady state selection [13,14].

Selection: Roulette Wheel

1. Better solutions get higher chance to become parents for next generation solutions
2. Roulette wheel technique:
    - Assign each individual part of the wheel
    - Spin wheel N times to select N individuals.
    - The Probability of selection of i th individual is:

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

where fi: fitness of i th individual, N: number of individuals. Figure 6 illustrate roulette wheel selection and figure 7 illustrate function select [10,15].

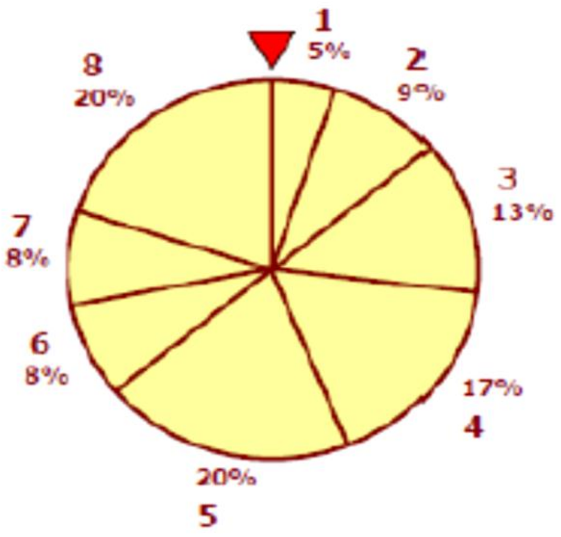

Figure 6. Illustrate roulette wheel selection.

```
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.

f. Crossover: There are a number of techniques, but all involve swapping genes—sequences of bits in the strings between two individuals (or between two strands of a diploid individual).

Crossover operator

1. Choose a random point.
2. Split parents at this crossover point.
3. Create children by exchanging tails.
4. Probability of crossover is typically in range (0.6, 0.9).

Figure 8 illustrate crossover operator [12].

Parent 1	1 1 0 1 $\boxed{0\ 0\ 1\ 0\ 1}$ 1 0 1 1 1 0 0
Parent 2	0 0 1 1 $\boxed{1\ 0\ 1\ 1\ 0}$ 1 0 0 1 0 1 1
Child	1 1 0 1 1 0 1 1 0 1 0 1 1 1 0 0

Figure 8. Crossover operator

Other Crossover Types

- Two-point crossover Figure 9 shows two point crossover operator.

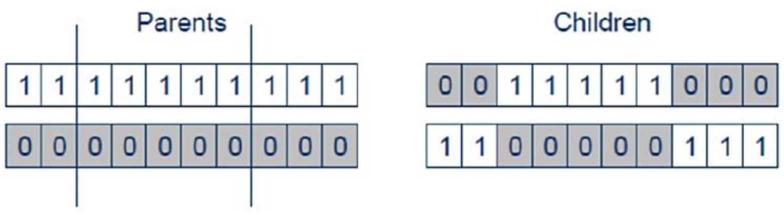

Figure 9. Two Point Crossover Operator.

- Uniform crossover: randomly generated mask Figure 10 shows uniform crossover operator.

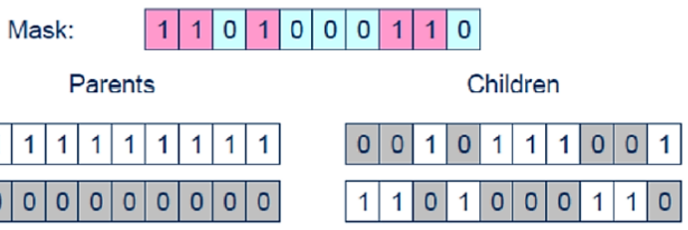

Figure 10. Uniform Crossover Operator.

Figure 11 illustrate procedure crossover [10,12].

```
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 11. Procedure crossover.

g. Mutation: Generally, bits are flipped with a small probability, this explores parts of the space that crossover might miss, and helps prevent premature convergence.

Mutation operator

1. Alter each gene independently
2. Mutation probability is typically in range (1/population size /chromosome length) [10,12].

Figure 12 illustrate mutation function operator.

Before 1 1 0 1 1 0 1 $\boxed{1}$ 0 1 0 1 1 1 0 0

After 1 1 0 1 1 0 1 $\boxed{0}$ 0 1 0 1 1 1 0 0

Figure 12. Mutation operator.

Figure 13 illustrate function mutation [10,12].

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

Figure 13. Function mutation.

h. Replacement:

1. Simple or generational GAs replace entire population per the dictates of the selection scheme
2. Steady state or online GAs use different replacement Scheme
    - Replace worst
    - Replace best
    - Replace parent
    - Replace random
    - Replace most similar (crowding)
3. Only replace worst and replace most similar are Generally very effective (when replace parent works, it is because parents are similar) [6,10]. Crowding algorithm shown in Figure 14 [3].

```
For 1 to crowding factor do
    x: = find worst of a random set
        If this is not more similar to offspring then past x then
            Set worst most similar to x
        End if
End for
   Replace worst most similar with offspring
```

Figure 14. Crowding Algorithm.

8. GENETIC ALGORITHM

In the genetic algorithm process is as follows:

Step 1. Determine the number of chromosomes, generation, and mutation rate and crossover rate value

Step 2. Generate chromosome-chromosome number of the population, and the initialization value of the genes chromosome-chromosome with a random value

Step 3. Process steps 4-7 until the number of generations is met

Step 4. Evaluation of fitness value of chromosomes by calculating objective function

Step 5. Chromosomes selection

Step 5. Crossover

Step 6. Mutation

Step 7. New Chromosomes (Offspring)

Step 8. Solution (Best Chromosomes) [5,10].

8. 1. Genetic Algorithm Code

Genetic algorithm code illustrated in Figure 15 [13.16,7].

```
begin
    t=0
    initialize $P_t$
    evaluate $P_t$
    while(termination condition not satisfied) do
    begin
        t=t+1
        select $P_t$ from $P_{t-1}$
        mutate $P_t$
        evaluate $P_t$
    end
end
```

Figure 15. Genetic algorithm Code.

8. 2. Genetic Algorithm Flowchart

The flowchart of genetic algorithm shown in Figure 16 [15,17].

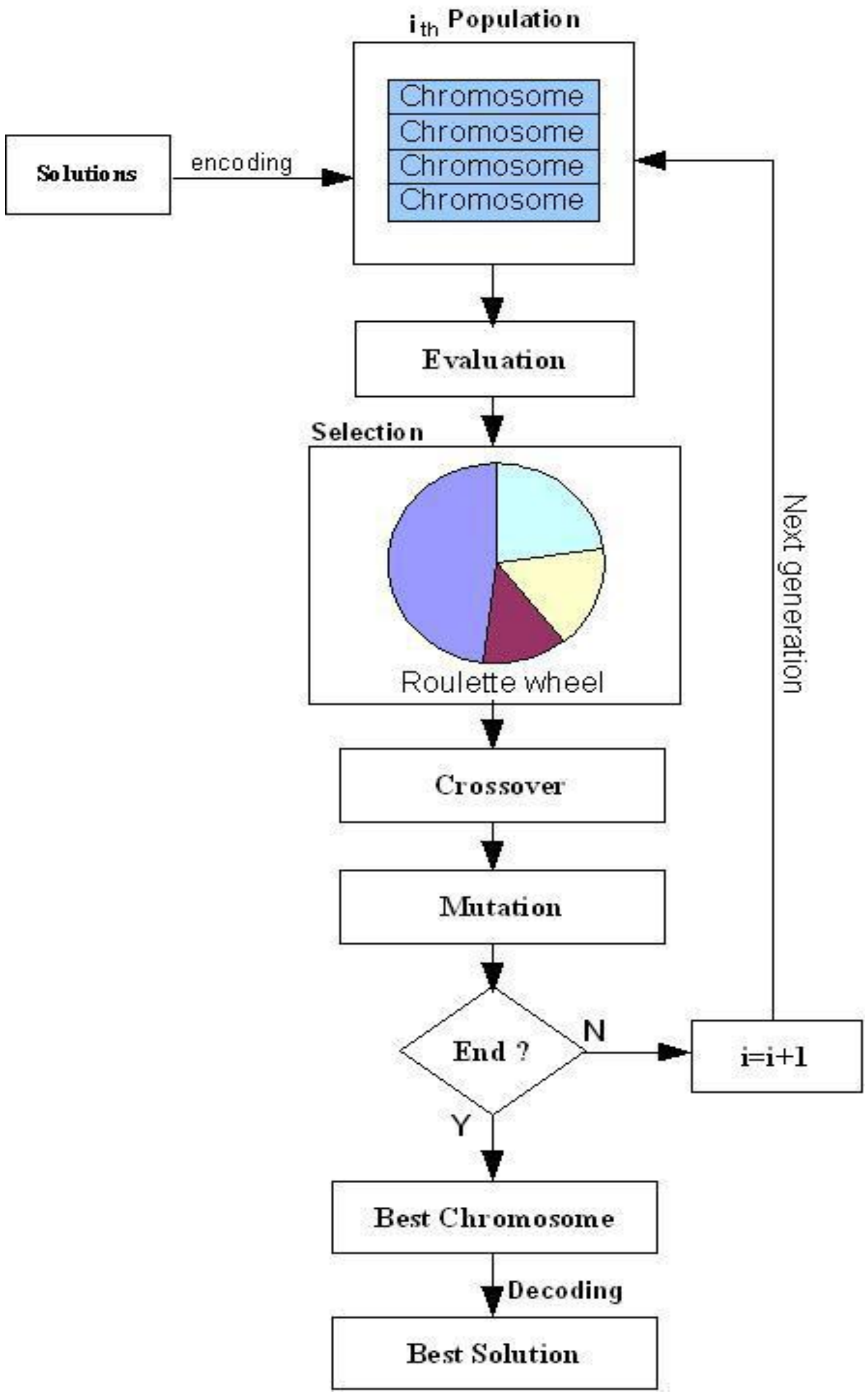

Figure 16. Genetic Algorithm Flowchart.

9. LINEAR EQUATION PROBLEM

In this work we use genetic algorithms to solve linear equation problem. Suppose there is equality a + 2 b + 3 c = 10, genetic algorithm will be used to find the value of a, b and c that satisfy the above equation. First we should formulate the objective function, for this problem the objective is minimizing the value of function f(x) where f (x) = ((a + 2b + 3c) – 10). To speed up the computation, we can restrict that the values of variables a, b, c, are integers between 0 and 10.

9. 1. Initialization

For example we define the number of chromosomes in population are 6, then we generate random value of gene a, b, c for 6 chromosomes

Chromosome[1] = [a;b;c] = [1;0;2]
Chromosome[2] = [a;b;c] = [2;2;3]
Chromosome[3] = [a;b;c] = [1;4;4]
Chromosome[4] = [a;b;c] = [2;1;6]
Chromosome[5] = [a;b;c] = [1;4;9]
Chromosome[6] = [a;b;c] = [2;5;2]

9. 2. Evaluation

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

F_obj[1] = Abs((1 + 2*0 + 3*2) - 10) =3
F_obj[2] = Abs((2 + 2*2 + 3*3) - 10) = 5
F_obj[3] = Abs((1 + 2*4 + 3*4) - 10) = 11
F_obj[4] = Abs((2 + 2*1 + 3*6) - 10) = 12
F_obj[5] = Abs((1 + 2*4 + 3*9) - 10) = 17
F_obj[6] = Abs((2 + 2*5 + 3*2) – 10 = 8

9. 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. To avoid divide by zero problem, the value of F_obj is added by 1.

Fitness[1] = 1 / (1+**F_obj**[1]) = 1/4 = 0.2500
Fitness[2] = 1 / (1+**F_obj**[2]) = 1/6 = 0.1666
Fitness[3] = 1 / (1+**F_obj**[3]) = 1/12 = 0.0833
Fitness[4] = 1 / (1+**F_obj**[4]) = 1/13 = 0.0769
Fitness[5] = 1 / (1+**F_obj**[5]) = 1/18 = 0.0555
Fitness[6] = 1 / (1+**F_obj**[6]) = 1/9 = 0.1111

Total = 0.25 + 0.1666 + 0.0833 + 0.0769 + 0.0555 + 0.1111 = 0.7434

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

P[1] = 0.2500 / 0.7434 = 0.3363
P[2] = 0.1666 / 0.7434 = 0.2241
P[3] = 0.0833 / 0.7434 = 0.1121
P[4] = 0.0769 / 0.7434 = 0.1034
P[5] = 0.0555 / 0.7434 = 0.0747
P[6] = 0.1111 / 0.7434 = 0.1494

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] = 0.3363
C[2] = 0.3363 + 0.2241 = 0.5604
C[3] = 0.3363 + 0.2241 + 0.1121 = 0.6725
C[4] = 0.3363 + 0.2241 + 0.1121 + 0.1034 = 0.7759
C[5] = 0.3363 + 0.2241 + 0.1121 + 0.1034 + 0.0747 = 0.8506
C[6] = 0.3363 + 0.2241 + 0.1121 + 0.1034 + 0.0747 + 0.1494 = 1.0000

Having calculated the cumulative probability of selection process using roulette-wheel can be done. The process is to generate random number R in the range 0-1 as follows.

R[1] = 0.390
R[2] = 0.173
R[3] = 0.988
R[4] = 0.711
R[5] = 0.287
R[6] = 0.490

If random number R [1] is greater than P [1] and smaller than P [2] then select Chromosome [2] as a chromosome in the new population for next generation:

NewChromosome[1] = **Chromosome**[2]
NewChromosome[2] = **Chromosome**[2]
NewChromosome[3] = **Chromosome**[4]
NewChromosome[4] = **Chromosome**[5]
NewChromosome[5] = **Chromosome**[6]
NewChromosome[6] = **Chromosome**[1]

Chromosome in the population thus became:

Chromosome[1] = [2;2;3]
Chromosome[2] = [2;2;3]
Chromosome[3] = [2;1;6]
Chromosome[4] = [1;4;9]
Chromosome[5] = [2;5;2]
Chromosome[6] = [1;0;2]

9. 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;
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] = 0.080
R[2] = 0.148
R[3] = 0.659
R[4] = 0.995
R[5] = 0.048
R[6] = 0.239

For random number R above, parents are Chromosome [1], Chromosome [4] and Chromosome [5] will be selected for crossover.

Chromosome[1] >< **Chromosome**[6]
Chromosome[2] >< **Chromosome**[5]
Chromosome[5] >< **Chromosome**[1]
Chromosome[6] >< **Chromosome**[2]

After chromosome selection, the next process is determining the position of the crossover point. This is done by generating random numbers between 1 to (length of Chromosome – 1). In this case, generated random numbers should be between 1 and 2. After we get the crossover point, parents Chromosome will be cut at crossover point and its gens will be interchanged. For example we generated 3 random number and we get:

C[1] = 1
C[2] = 1
C[3] = 1
C[4] = 1

Then for crossover, crossover, parent’s gens will be cut at gen number 1, e.g.

Chromosome[1] >< **Chromosome**[6]
= [2;2;3] >< [1;0;2]
= [2;0;2]
Chromosome[2] >< **Chromosome**[5]
= [2;2;3] >< [2;5;2]
= [2;5;2]
Chromosome[5] >< **Chromosome**[1]
= [2;5;2] >< [2;2;3]
= [2;2;3]
Chromosome[6] >< **Chromosome**[2]
= [1;0;2] >< [2;2;3]
= [1;2;3]

Thus Chromosome population after experiencing a crossover process:

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

9. 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 = 3*6 = 18

Mutation process is done by generating a random integer between 1 and total gen (1 to 18). 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 * 18 = 1.8 ≈ 2

Suppose generation of random number yield 10 and 14 then the chromosome which have mutation are Chromosome number 4 gen number 1 and Chromosome 5 gen number 2. The value of mutated gens at mutation point is replaced by random number between 0-10. Suppose generated random number are 1 and 0 then Chromosome composition after mutation are:

Chromosome[1] = [2;2;3]
Chromosome[2] = [2;2;3]
Chromosome[3] = [2;0;2]
Chromosome[4] = [**1**;5;2]
Chromosome[5] = [1;**0**;3]
Chromosome[6] = [1;0;2]

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:

Chromosome[1] = [2;2;3]
F_obj[1] = Abs((2 + 2*2 + 3*3) - 10) = 5
Chromosome[2] = [2;2;3]
F_obj[2] = Abs((2 + 2*2 + 3*3) - 10) = 5
Chromosome[3] = [2;0;2]
F_obj[3] = Abs((2 + 2*0 + 3*2) - 10) = 2
Chromosome[4] = [1;5;2]
F_obj[4] = Abs((1 + 2*5 + 3*2) - 10) = 7
Chromosome[5] = [1;0;3]
F_obj[5] = Abs((1 + 2*0 + 3*3) - 10) = 0
Chromosome[6] = [1;0;2]
F_obj[6] = Abs((1 + 2*0 + 3*2) - 10) = 3

From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous Chromosome generation.

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 example, after running 100 generations, best chromosome is obtained:

Chromosome = [1; 0; 3]

This means that:

a = 1, b = 0, c = 3

If we use the number in the problem equation

a +2 b +3 c = 10
1 + (2 * 0) + (3 * 3) = 10

We can see that the value of variable a, b and c generated by genetic algorithm can satisfy that equality.

10. EXPERIMENTAL RESULT

Executing the simple genetic algorithm code (SGA) written in Pascal programming language, The primary work of the SGA is performed in three routines, select crossover, and mutation As illustrated in Figures 1,2,3 respectively. Their action is coordinate by a procedure called generation that generates a new population at each successive generation. At the end of the run repeating the process hundreds of times until the best solution is determined. The primary data structure is the population of 14 string and the following parameters:

SGA Parameters
Population size = **14**
Chromosome length = **3**
Maximum of generation = **10**
Crossover probability = **0.25**
Mutation probability = **0.01**

Linear equation Population

A value	B value	C value
0	2	2
1	0	3
4	0	2
7	0	1
0	5	0
2	4	0
4	3	0
6	2	0
8	1	0
10	0	0
5	1	1
3	2	1
1	3	1
2	1	2

11. CONCLUSIONS

- Objective function 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 objective 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.
- Genetic algorithms are a very powerful tool, allowing searching for solutions when there are no other feasible means to do so.

- The algorithm is easy to produce and simple to understand and enhancements easily introduced. This gives the algorithm much flexibility.
- 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 linear equation.

References

[1] Bashkansky Guy and Yaari, Yaakov, ”Black Box Approach for Selecting Optimization Options Using Budget-Limited Genetic Algorithms”, 2007.

[2] Ms. Dharmistha, D. Vishwakarma, ”Genetic Algorithm based Weights Optimization of Artificial Neural Network”, *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 2012.

[3] Goldberg David E, “Genetic Algorithm in Search, Optimization, and Machine Learning”, Addison Wesley Longmont, International Student Edition, 1989.

[4] Popov Andrey, ”Genetic algorithms for optimization”, Hamburg, 2005.

[5] Zhu Nanhao and O’Connor Ian, ” iMASKO: A Genetic Algorithm Based Optimization Framework for Wireless Sensor Networks”, *Journal of Sensor and Actuator Networks* ISSN 2224-2708, www.mdpi.com/journal/jsan/ 2013.

[6] Yaeger Larry, ”Intro to Genetic Algorithms”Artificial Life as an approach to Artificial Intelligence”Professor of Informatics, Indiana University, 2008.

[7] Lubna Zaghlul Bashir, ”Find Optimal Solution For Eggcrate Function Based On Objective Function”, *World Scientific News*, 10 (2015) 53-72.

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

[9] Lubna Zaghlul Bashir, Nada Mahdi, ”Use Genetic Algorithm in Optimization Function For Solving Queens Problem”, *World Scientific News*, 5 (2015) 138-150.

[10] Peshko Olesya, “Global Optimization Genetic Algorithms”. 2007.

[11] Lubna Zaghlul Bashir, Raja Salih,“ Solving Banana (Rosenbrock) Function based on fitness function”, *World Scientific News*, 6 (2015) 41-56.

[12] Lubna Zaghlul Bashir, ”Using Evolutionary Algorithm to Solve Amazing Problem by two ways: Coordinates and Locations”, *World Scientific News*, 7 (2015) 66-87.

[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] Hasan Saif, Chordia Sagar, Varshneya Rahul, ”Genetic algorithm”, 2012.

[16] Reeves .Colin,”genetic algorithms”, School of Mathematical and Information Sciences Coventry University Priory St Coventry CV1 5FBE-mail: C. Reeves@conventry. ac.uk http://www.mis.coventry.ac.uk/~colinr, 2005.

[17] Ghosh Ashish, Dehuri Satchidananda, ”Evolutionary algorithms for multi criterian optimization: a servay”, *International journal of computing and information system,* 2004.

(Received 15 August 2015; accepted 01 September 2015)