

OPTIMIZING THE POLYNOMIAL USING GENETIC ALGORITHM

A Project Report

Submitted in Partial Fulfillment of the Requirement for the Award of the Degree of

**BACHELOR OF TECHNOLOGY
(COMPUTER SCIENCE & ENGINEERING)**

To



**VEER BAHADUR SINGH PURVANCHAL UNIVERSITY,
JAUNPUR**

Under the Supervision of

Mr. Dileep Kumar Yadav
Assistant Professor

Submitted By:

Ashwani Patel (185512)

Ankit Gwal (185525)

Aryan Singh (185541)



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNSIET VBSPU JAUNPUR**

August, 2022

CANDIDATE'S DECLARATION

We, Ashwani Patel(185512), Ankit Gwal(185525), Aryan Singh(185541) are the student of B.Tech(Computer Science & Engineering) at **Uma Nath Singh Institute of Engineering and Technology, VBS Purvanchal University, Jaunpur**, declare that the work presented in this project titled “**Optimizing The Polynomial By Using Genetic Algorithm**”, submitted to **Department of Computer Science & Engineering** for the award of Bachelor of Technology degree in Computer Science & Engineering. All the work done in this project is entirely our own except for the reference quoted. To the best of our knowledge, this work has not been submitted to any other university or Institution for award of any degree.

Date: 29-07-2022

Place: UNSIET, Jaunpur

Student's Name

Ashwani Patel (185512)

Ankit Gwal (185525)

Aryan Singh (185541)

B.Tech(CSE) 4th Year

CERTIFICATE

It is certified that, this project entitled “**Optimizing The Polynomial By Using Genetic Algorithm**”, submitted by (Ashwani Patel, Ankit Gwal, Aryan Singh) in partial fulfillment of the requirement for the award of Bachelor of Technology in computer science & Engineering degree from VBS Purvanchal University, Jaunpur, is record of students own study carried under my supervision. This Project report has not been submitted to any other university or institution for the award of any degree.

Project Guide

Mr. Dileep Kumar Yadav
Assistant Professor
Department of CSE

External Examiner

Date:
Place: UNSIET, Jaunpur

ABSTRACT

Genetic algorithm is available for optimizing the static polynomial but if we want dynamic polynomial, means polynomial given at run time of program then updation in the algorithm is needed. Working and implementing a major on actual environment needs system chunking, that means dividing project as the team member's calibre of seeing project as their domain skills. First of all we studied the concepts regarding genetic algorithm from different sources specially papers on IEEE, Wikipedia, Geeks for Geeks etc. Secondly for implementing genetic algorithm in Python language, modules required for it , are studied deeply. Thirdly we took a polynomial and done the dry run of the algorithm on that polynomial on the paper and verify the result manually. Finally, We took the polynomial and applied the concepts whatever we had studied in genetic algorithm and wrote the complete algorithm and done the execution process on the IDLE shell. We have received the "Successful" Program Execution as an acknowledgement.

ACKNOWLEDGEMENT

We are greatly indebted to **Mr. Dileep Kumar Yadav** (Assistant Professor, Dept. of CSE & IT) for his guidance and valuable advice that helped us in the preparation of this paper.

This work is not just an individual contribution till its completion. We would like to express our sincere gratitude to **Dr. Sanjeev Gangwar**, Head of Department of Computer Science & Engineering for his support to shape this paper in systematic way. We specially thanks to all faculty members who has given us this golden opportunity to work in new area. The satisfaction that accompanies that the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible, whose constant guidance and encouragement crown all efforts with success.

At last we must express our sincere heartfelt gratitude to all the staff members of Computer Science & Engineering department who helped us directly or indirectly during this course of work.

Student'S Name

Ashwani Patel (185512)

Ankit Gwal (185525)

Aryan Singh (185541)

B.Tech CSE 4th Year

TABLE OF CONTENTS

CONTENTS	PAGE NO.
Candidate Declaration	i
Certificate	ii
Abstract	iii
Acknowledgement	iv
List of Figures	viii
List of Tables	ix
CHAPTER 1	1
1. INTRODUCTION	1
1.1 Background	1
1.2 Foundation of Genetic Algorithm	2
1.3 Search Space	2
1.4 Fitness Score	3
1.5 Nature to Computer Mapping	5
1.6 Example Problem and Solution Using Genetic Algorithm	6
1.7 Advantages of Genetic Algorithm	6
1.8 Limitations of Genetic Algorithm	6
1.9 Application of Genetic Algorithms	7
1.9.1 Optimization	7
1.9.2 Economics	7
1.9.3 Neural Networks	7
1.9.4 Parallelization	7
1.9.5 Image Processing	7
1.9.6 Vehicle routing problems	7
1.9.7 Scheduling applications	7
1.9.8 Machine Learning	7
1.9.9 Robot Trajectory Generation	7
1.9.10 Parametric Design of Aircraft	8
1.9.11 DNA Analysis	8

1.9.12 Multimodal Optimization	8
1.9.13 Traveling salesman problem and its applications	8
CHAPTER 2	9
2. PROBLEM FORMULATION AND PROPOSED WORK	9
2.1 Problem Definition	9
2.2 Objectives	9
2.3 Proposed Work	10
CHAPTER 3	11
3. METHODOLOGY	11
3.1 Implementation Strategy	11
3.1.1 Flow-Chart Of Genetic Algorithm	11
3.1.1.1 Step 1 Initial Population	12
3.1.1.2 Step 2 Selection Of The Fittest Parent	13
3.1.1.3 Step 3 Crossover Operations	15
3.1.1.4 Step 4 Mutation Operations	16
3.1.2 Algorithm	18
3.2 Limitation	19
3.3 Tools/Hardware/Software to Be Used	20
3.3.1. NumPy Module	20
3.3.1.1 Operations using NumPy	21
3.3.1.2 Methods In Numpy	22
3.3.2 Random Module	22
3.3.2.1 Generate Random Floats	22
3.3.2.2 Generate Random Integers	22
3.3.2.3 Generate Random Numbers within Range	23
3.3.2.4 Select Random Elements	23
3.3.2.5 Shuffle Elements Randomly	23
3.4 Expected Outcome (Performance Metrics with Details)	24
3.4.1 Initial Population	24
3.4.2 Fitness Score Calculation	25
3.4.3 Selection of Fittest Parents	26

3.4.4 Single-Point Crossover	27
3.4.5 Mutation	28
3.4.6 Iteration of Generation	29
CONCLUSION	30
REFERENCES	31

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
Fig.1.1.	Optimization	2
Fig.1.2.	Initial Population	3
Fig.1.3.	Fitness Function	4
Fig.3.1.	Flow-chart	11
Fig.3.2.	Initial Population	12
Fig.3.3.	Roulette Wheel Selection	13
Fig.3.4	Tournament Selection	14
Fig.3.5.	Crossover	15
Fig.3.6.	One Point Crossover	15
Fig.3.7.	Multi Point Crossover	16
Fig.3.8.	Uniform Crossover	16
Fig.3.9.	Mutation	17
Fig.3.10.	Numpy Module	21
Fig.3.11.	Initial Population	24
Fig.3.12.	Fitness Score Calculation	25
Fig.3.13.	Selection Of Fittest Parents	26
Fig.3.14.	Single-Point Crossover	27
Fig.3.15.	Mutation	28
Fig.3.16.	Iteration of generation	29

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
Table 1.1.	Natural to Computer Mapping	5

CHAPTER 1

1. INTRODUCTION

1.1 BACKGROUND

Genetic Algorithm was developed by John Holland and his collaborator in 1960. GA is a model or abstraction of biological evolution based on theory of natural selection. To solve an optimization problem, GA uses the value of objective (fitness or merit) function assigned to the problem. It can deal with any type of optimization problem where the objective function may be stationary, non-stationary, differentiable, non-differentiable, linear, nonlinear, continuous, discontinuous, or having random noise.

Genetic Algorithms (GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to next generation. In simple words, they simulate “survival of the fittest” among individual of consecutive generation for solving a problem. Each generation consist of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

Optimization is the process of making something better. Optimization refers to finding the values of inputs in such a way that we get the “best” output values. The definition of “best” varies from problem to problem, but in mathematical terms, it refers to maximizing or minimizing one or more objective functions, by varying the input parameters.

The set of all possible solutions or values which the inputs can take make up the search space. In this search space, lies a point or a set of points which gives the optimal solution. The aim of optimization is to find that point or set of points in the search space.

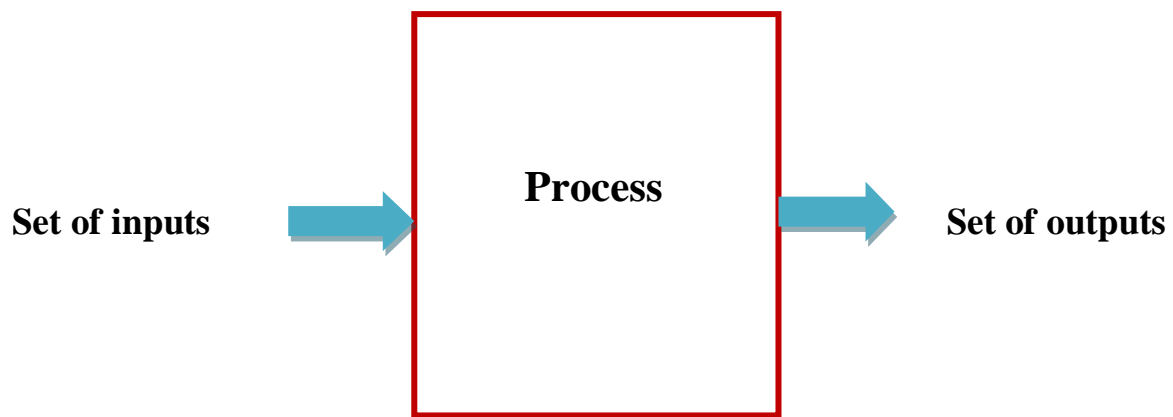


Fig. 1.1. Optimization

1.2 FOUNDATION OF GENETIC ALGORITHM

Genetic algorithms are based on an analogy with genetic structure and behaviour of chromosomes of the population.

Following is the foundation of GAs based on this analogy.

- Individual in population compete for resources and mate
- Those individuals who are successful (fittest) then mate to create more offspring than others.
- Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
- Thus each successive generation is more suited for their environment.

1.3 SEARCH SPACE

The population of individuals are maintained within search space. Each individual represents a solution in search space for given problem. Each individual is coded as a finite length vector (analogous to chromosome) of components. These variable components are analogous to Genes. Thus a chromosome (individual) is composed of several genes (variable components).

If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called search space (also state space). Each point in the search space represents one feasible solution. Each feasible solution can be "marked" by its value or

fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space. The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.

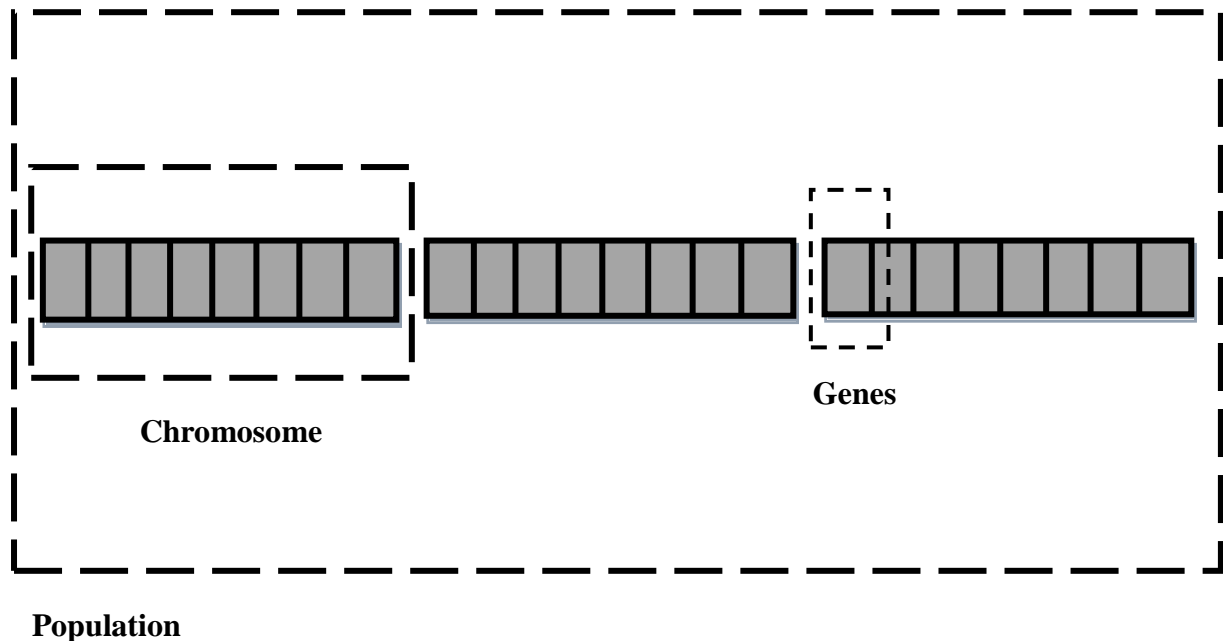


Fig.1.2. Initial Population

1.4 FITNESS SCORE

A Fitness Score is given to each individual which shows the ability of an individual to “compete”. The individual having optimal fitness score (or near optimal) are sought. The GAs maintains the population of n individuals (chromosome/solutions) along with their fitness score. The individuals having better fitness scores are given more chance to reproduce than others. The individuals with better fitness scores are selected who mate and produce better offspring by combining chromosomes of parents. The population size is static so the room has to be created for new arrivals. So, some individuals die and get replaced by new arrivals eventually creating new generation when all the mating opportunity of the old population is exhausted. It is hoped that over successive generations better solutions will arrive while least fit die. Each new generation has on average more “better genes” than the individual (solution) of previous generations. Thus each new generation have better “partial solutions” than previous generations. Once the offspring produced having no significant

difference from offspring produced by previous populations, the population is converged. The algorithm is said to be converged to a set of solutions for the problem.

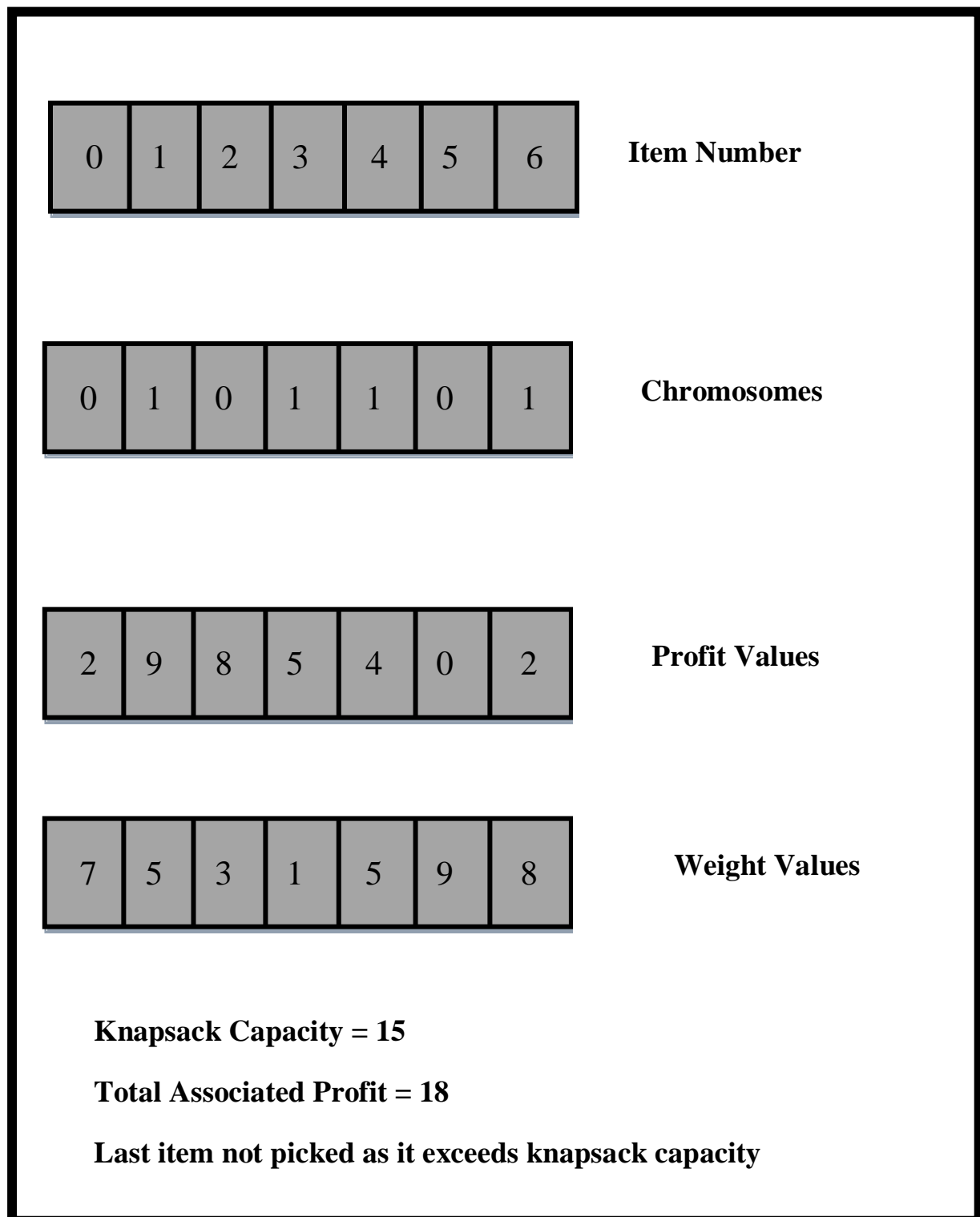


Fig.1.3. Fitness Function

1.5 NATURE TO COMPUTER MAPPING

Nature	Computer
Population	Set of solutions
Individual	Solution to a problem
Fitness	Quality of a solution
Chromosome	Encoding for a solution
Gene	Part of the encoding solution

Table 1.1. Natural to Computer Mapping

1.6 EXAMPLE PROBLEM AND SOLUTION USING GENETIC ALGORITHM

Given a target string, the goal is to produce target string starting from a random string of the same length. In the following implementation, following analogies are made –

- Characters A-Z, a-z, 0-9, and other special symbols are considered as genes.
- Fitness score is the number of characters which differ from characters in target string at a particular index. So individual having lower fitness value is given more preference.
- A string generated by these characters is considered as a chromosomes or solutions or individuals.

1.7 ADVANTAGES OF GENETIC ALGORITHM

GAs have various advantages which have made them immensely popular. These include

- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of “good” solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

1.8 LIMITATIONS OF GENETIC ALGORITHM

Like any technique, GAs also suffer from a few limitations. These include

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.

- If not implemented properly, the GA may not converge to the optimal solution.

1.9 APPLICATION OF GENETIC ALGORITHMS

Genetic Algorithms are primarily used in optimization problems of various kinds, but they are frequently used in other application areas as well.

In this section, we list some of the areas in which Genetic Algorithms are frequently used.

1.9.1 Optimization

Genetic Algorithms are most commonly used in optimization problems wherein we have to maximize or minimize a given objective function value under a given set of constraints. The approach to solve Optimization problems has been highlighted throughout the tutorial.

1.9.2 Economics

GAs are also used to characterize various economic models like the cobweb model, game theory equilibrium resolution, asset pricing, etc.

1.9.3 Neural Networks

GAs are also used to train neural networks, particularly recurrent neural networks.

1.9.4 Parallelization

GAs also have very good parallel capabilities, and prove to be very effective means in solving certain problems, and also provide a good area for research.

1.9.5 Image Processing

GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.

1.9.6 Vehicle routing problems

With multiple soft time windows, multiple depots and a heterogeneous fleet.

1.9.7 Scheduling applications

GAs are used to solve various scheduling problems as well, particularly the time tabling problem.

1.9.8 Machine Learning

As already discussed, genetics based machine learning (GBML) is a niche area in machine learning.

1.9.9 Robot Trajectory Generation

GAs have been used to plan the path which a robot arm takes by moving from one point to another.

1.9.10 Parametric Design of Aircraft

GAs have been used to design aircrafts by varying the parameters and evolving better solutions.

1.9.11 DNA Analysis

GAs have been used to determine the structure of DNA using spectrometric data about the sample.

1.9.12 Multimodal Optimization

GAs are obviously very good approaches for multimodal optimization in which we have to find multiple optimum solutions.

1.9.13 Traveling salesman problem and its applications

GAs have been used to solve the TSP, which is a well-known combinatorial problem using novel crossover and packing strategies.

CHAPTER 2

2. PROBLEM FORMULATION AND PROPOSED WORK

2.1 PROBLEM DEFINITION

There is given a polynomial of any degree. We are having few values on which polynomial will respond with some value which will be generated in our program with the use of random method. We need to find the new and most optimal value with use of the previous initial value which are generated with the help of random method. For this we are going to use genetic algorithm approach which will solve the problem genetically and will respond in best optimal solution of given polynomial with use of initial values. Problem as per for sight looks little easy but as approach used is going to look little typical but genetic algorithm serialised the steps performs the problem in fewer steps and generates most optimal solution of given polynomial.

2.2 OBJECTIVES

The main objective of this project was to make use of genetic algorithm to find the most optimal solution of polynomial. In this process, first we need to generate some of the initial population for generating fittest offspring. To generate the fittest offspring we need to perform mating operation between two highest fittest parents. To make the diversity in the offspring we need to perform some of the reproduction mechanism that is, Crossover and mutation. Based on the deep analysis of the algorithm, we concluded it can be said that taking help of some random values of solution of polynomial, new values can be generated which will give most optimal solution for polynomial than previously selected values.

2.3 PROPOSED WORK

First of all we studied the concepts regarding genetic algorithm from different sources specially papers on IEEE, Wikipedia, Geeks for Geeks. Secondly for implementing genetic algorithm in Python language, modules required for it, are studied deeply. Thirdly we took a polynomial and done the dry run of the algorithm on that polynomial on the paper and verify the result manually. Finally, We took the polynomial and applied the concepts whatever we have studied in genetic algorithm and wrote the complete algorithm and done the execution process on the IDLE shell. We have received “the successful” program execution as an acknowledgement.

CHAPTER 3

3. METHODOLOGY

3.1 IMPLEMENTATION STRATEGY

3.1.1 Flow-Chart Of Genetic Algorithm

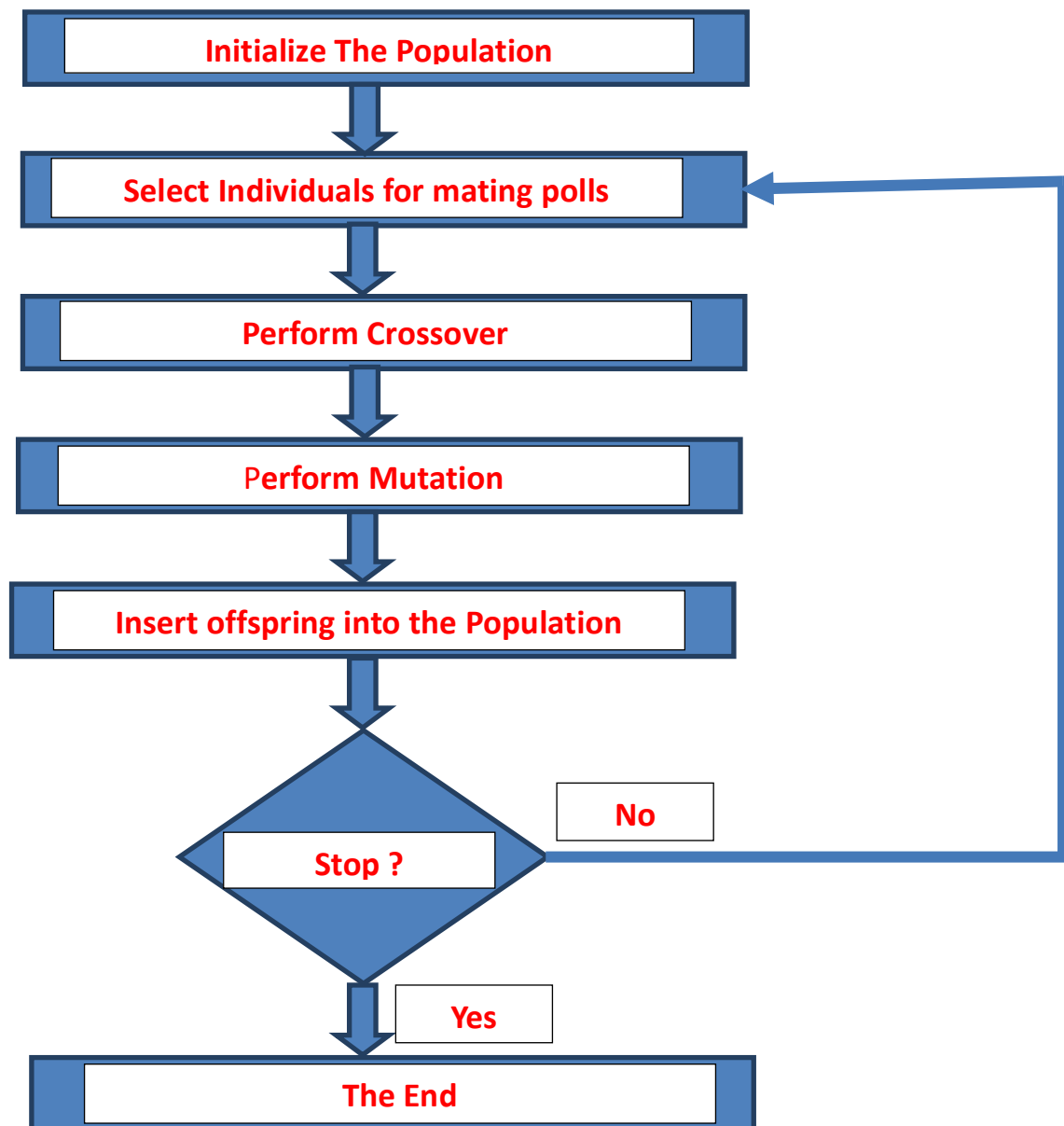


Fig.3.1. Flow-chart

3.1.1.1 Step 1 Initial Population

First of all we have to take some of the population means random values of and find the fitness score to do the offspring generation process. After this we have to take two fittest parents and do the mating to generate best offspring. If we got the best offspring with the highest fitness score then we stop the next offspring generation process and terminate from the generation loop and return the fittest offspring which is optimum. There are two primary methods to initialize a population in a GA.

Random Initialization populates the initial population with completely random solutions.

Heuristic Initialization populates the initial population using a known heuristic for the problem.

In research, it has been observed that

- When the entire population is initialized using Heuristic initialization, it can result in the population having similar solutions and very little diversity.
- Random solutions are the ones to drive the population to optimality.
- Heuristic initialization affects the initial fitness of the population.
- It is the diversity of the solutions which lead to optimality
- Therefore, the best practice is starting with heuristic initialization, just seeding the population with some initial good solutions and then filling up the rest with random solutions.



Fig.3.2. Initial Population

3.1.1.2 Step 2 Selection Of The Fittest Parent

In this step ,The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations. The process that determines which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. The primary objective of the selection operator is to emphasize the good solutions and eliminate the bad solutions in a population while keeping the population size constant. “Selects the best, discards the rest”.

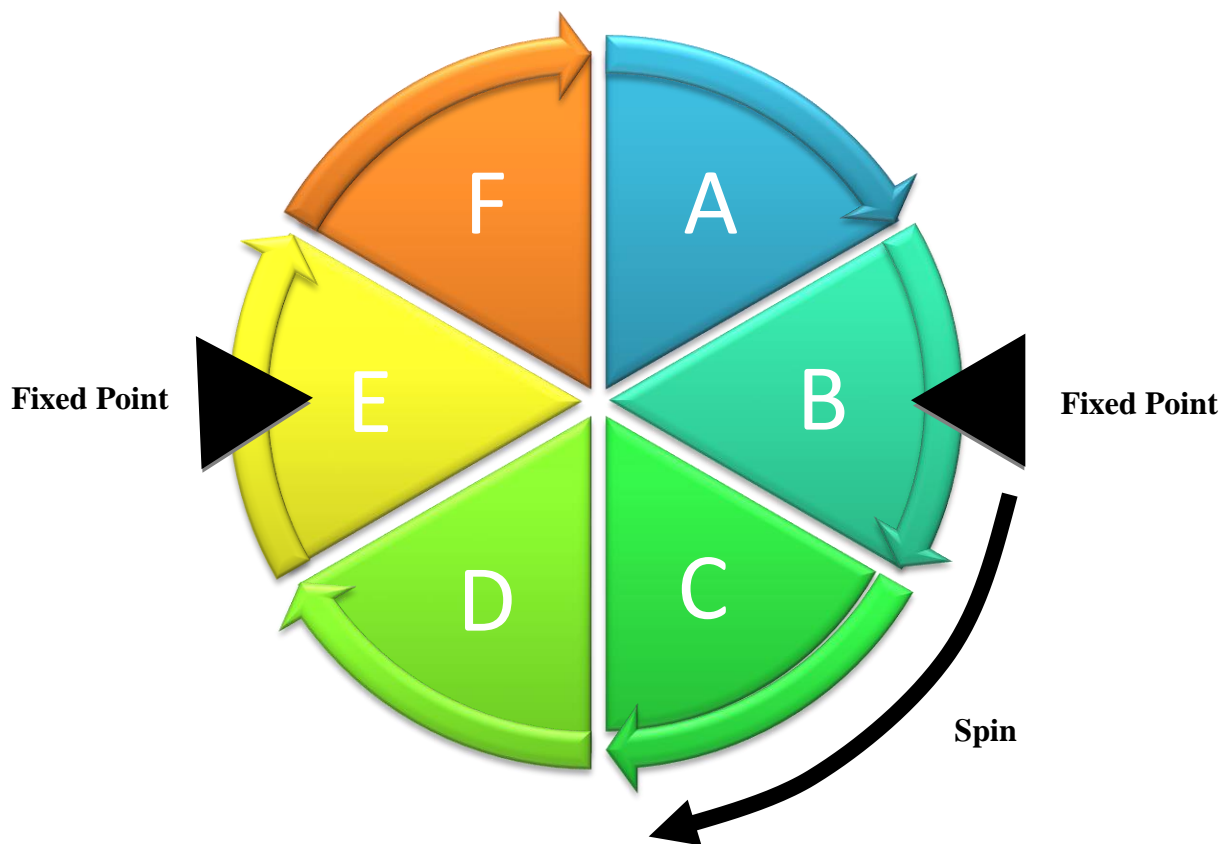


Fig.3.3. Roulette Wheel Selection

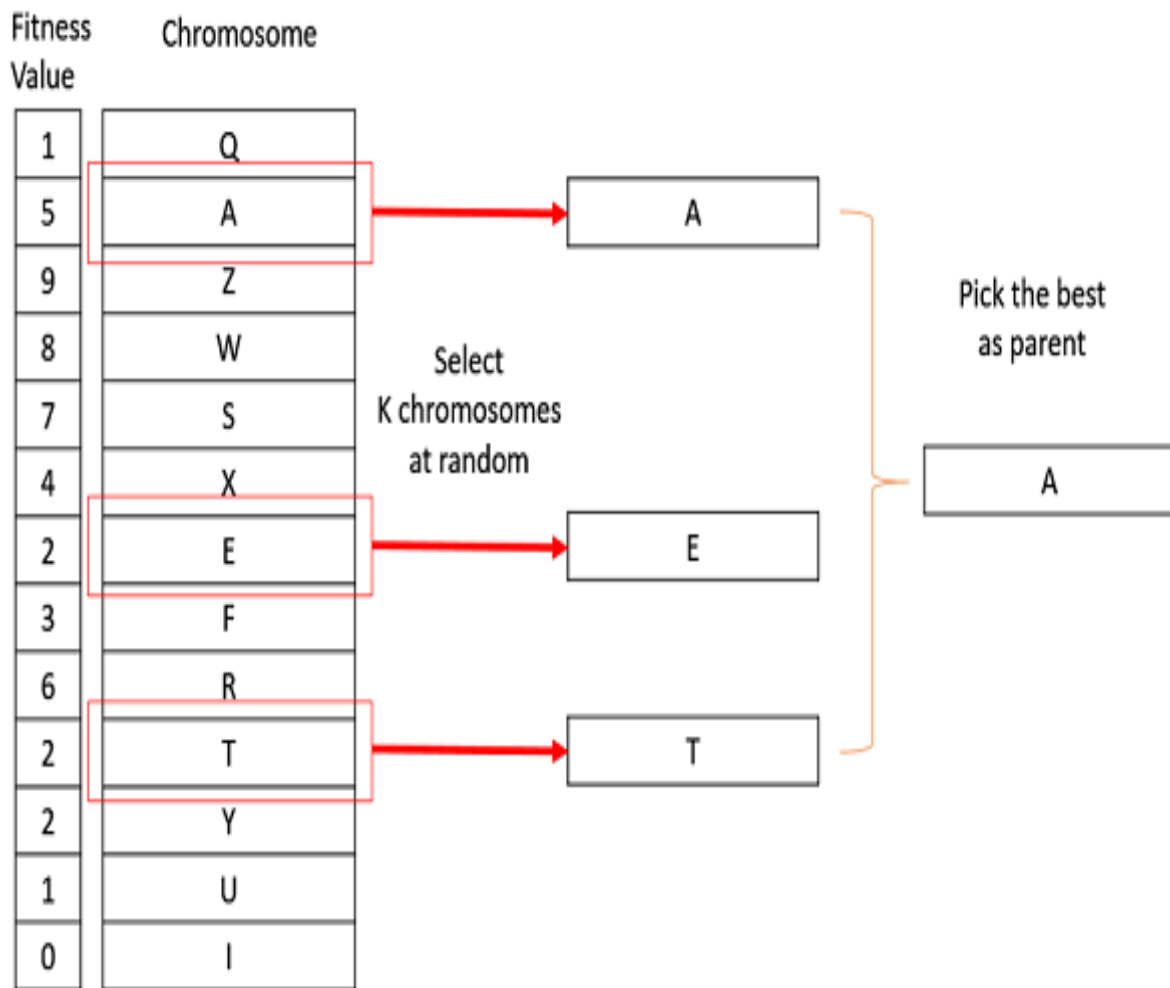


Fig.3.4. Tournament Selection

In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.

3.1.1.3 Step 3 Crossover Operations

This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

For example

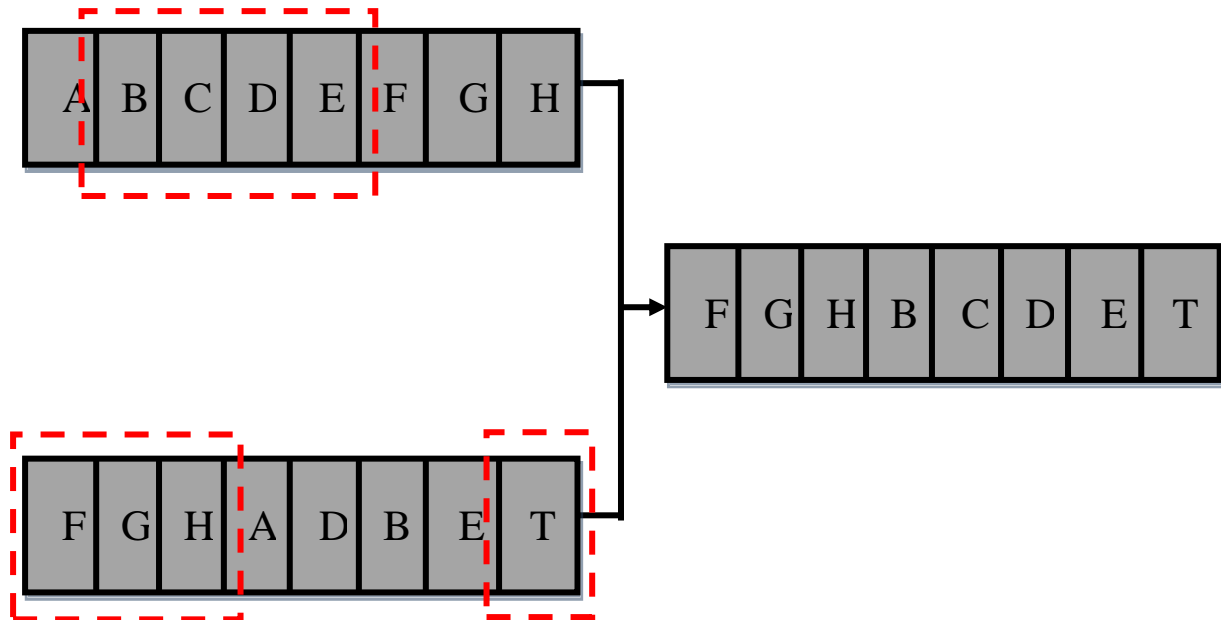


Fig.3.5. Crossover

One-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.

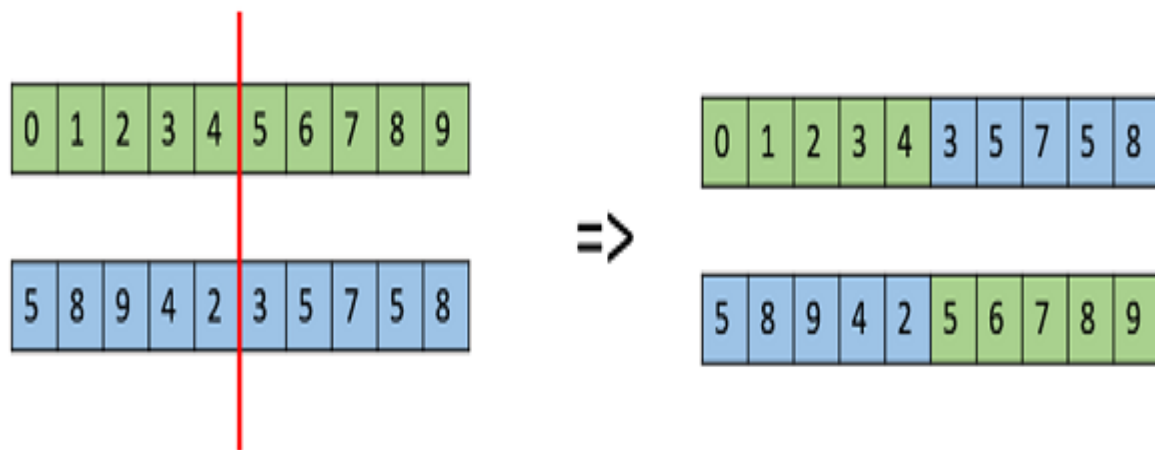


Fig.3.6. One Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.

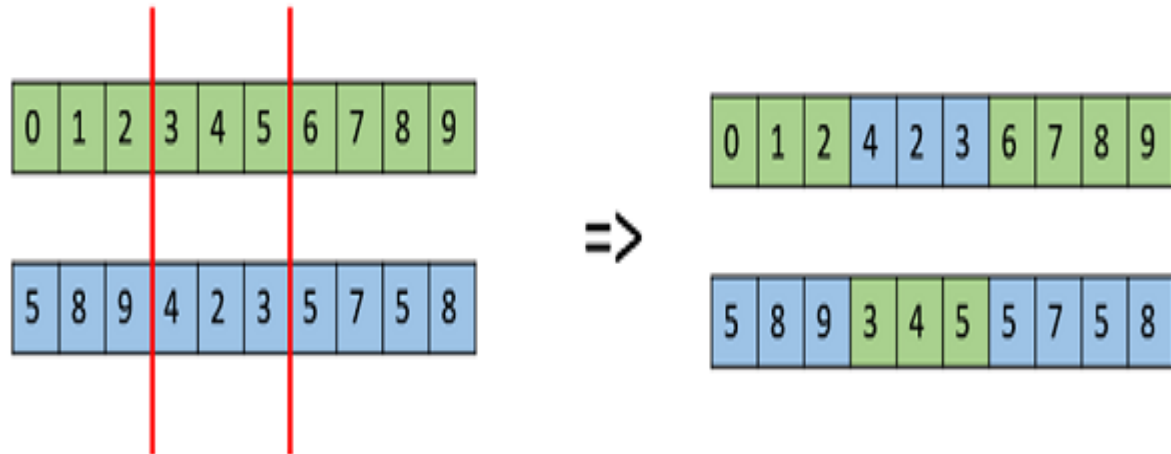


Fig.3.7. Multi Point Crossover

Uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

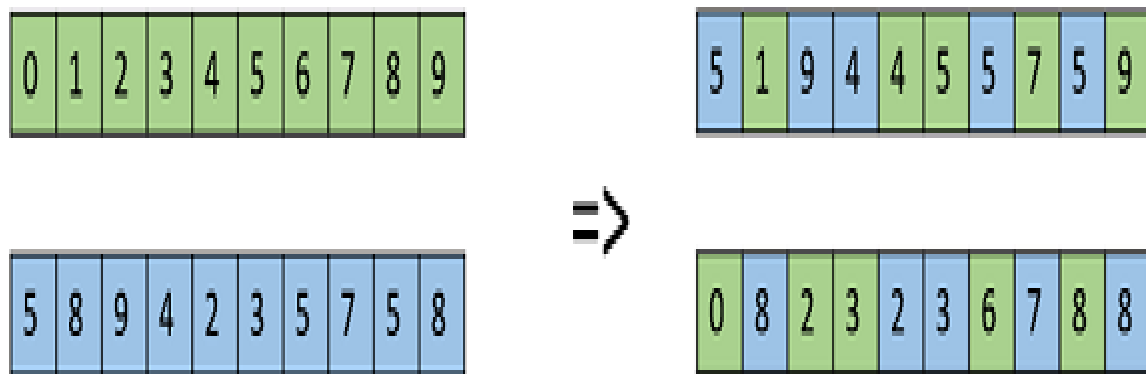


Fig.3.8. Uniform Crossover

3.1.1.4 Step 4 Mutation Operations

The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

For example

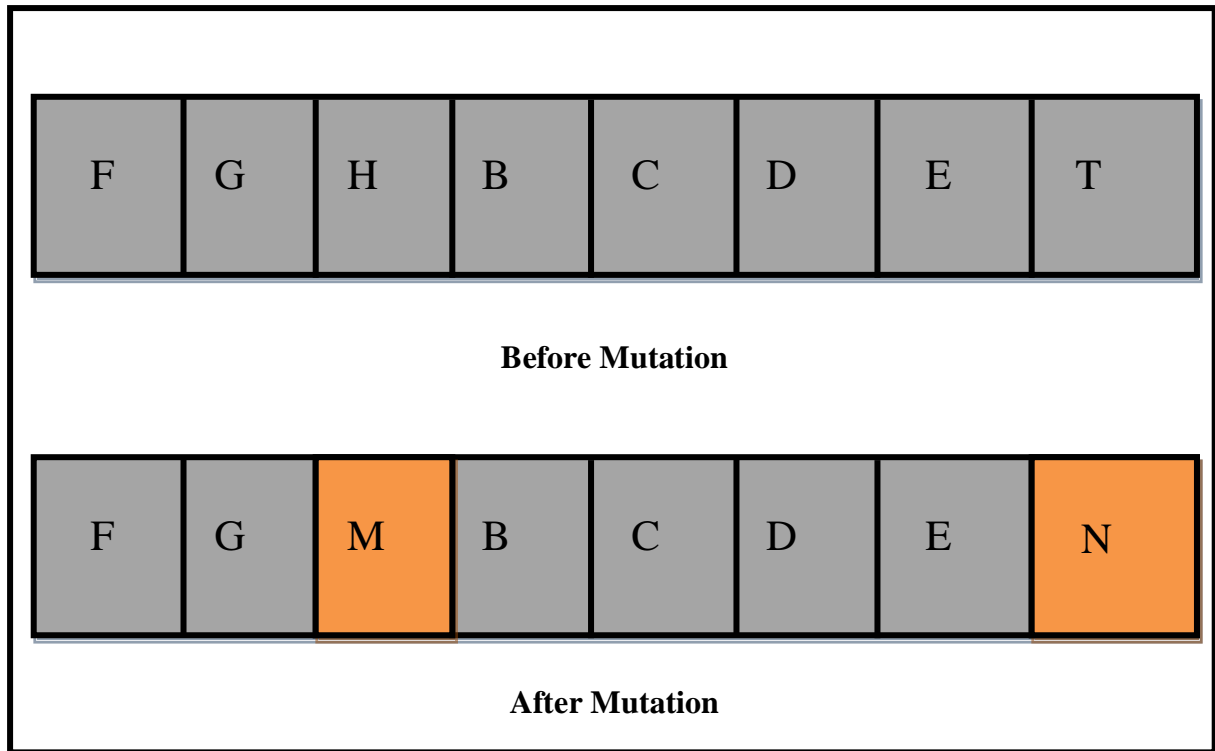


Fig.3.9. Mutation

3.1.2 Algorithm

The whole algorithm can be summarized as given pseudo code

Randomly initialize population p

Determine fitness of population

Until convergence repeat

 Select parents from population.

 Crossover and generate new population.

 Perform mutation on new population.

 Calculate fitness for new population.

Another Way to Represent the Algorithm

Function GA () {

 Initialize population;

 Calculate fitness function ();

 While (fitness value! = termination criteria)

 {

 Selection ();

 Crossover ();

 Mutation ();

 Calculate fitness function ();

 }

 }

3.2 LIMITATION

There are some limitations of the use of a genetic algorithm compared to alternative optimization algorithms:

- Repeated fitness function evaluation for complex problem is often the most prohibitive and limiting segment of artificial evolutionary algorithms. Finding the optimal solution to complex high-dimensional, multimodal problems often requires very expensive fitness function evaluations.
- Genetic algorithms do not scale well with complexity. That is, where the number of elements which are exposed to mutation is large there is often an exponential increase in search space size. This makes it extremely difficult to use the technique on problems such as designing an engine, a house or a plane.
- The "better" solution is only in comparison to other solutions. As a result, the stop criterion is not clear in every problem.
- Operating on dynamic data sets is difficult, as genomes begin to converge early on towards solutions which may no longer be valid for later data. Several methods have been proposed to remedy this by increasing genetic diversity somehow and preventing early convergence, either by increasing the probability of mutation when the solution quality drops (called triggered hyper permutation), or by occasionally introducing entirely new, randomly generated elements into the gene pool (called random immigrants).

3.3 TOOLS/HARDWARE/SOFTWARE TO BE USED

- Generating algorithm related content on web browser
- Research Paper On IEEE
- PYTHON Programming Language With some popular module of python as Numpy.
- Our team work
- Processor 64-bit, four-core, 2.5 GHz minimum per core
- RAM MINIMUM 4GB
- Hard disk Greater than 100GB
- Operating system windows OS
- Minimum CPU or processor speed
- Minimum GPU or video memory.
- Minimum system memory (RAM)
- Minimum free storage space
- Audio hardware (sound card, speakers, etc)

3.3.1. NumPy Module

- NumPy is a Python library used for working with arrays.
- It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- NumPy stands for Numerical Python.
- NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently. This behavior is called locality of reference in computer science.
- This is the main reason why NumPy is faster than lists. Also it is optimized to work with latest CPU architectures.
- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

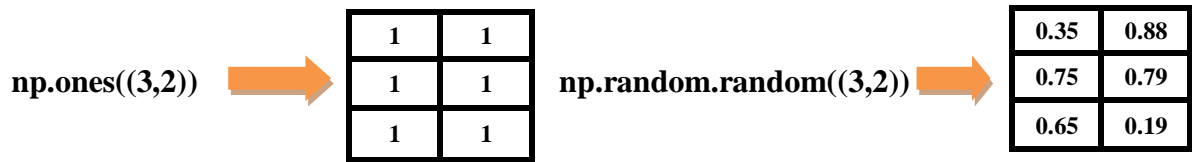


Fig.3.10. Numpy Module

3.3.1.1 Operations using NumPy

Using NumPy, a developer can perform the following operations

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.
- NumPy A Replacement for MatLab.
- It is open-source, which is an added advantage of NumPy.
- The most important object defined in NumPy is an N-dimensional array type called ndarray. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.
- Every item in a ndarray takes the same size as the block in the memory. Each element in ndarray is an object of the data-type object (called dtype).
- Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types. The following diagram shows a relationship between ndarray, data-type object (dtype) and array scalar type
- An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows

Take a look at the following examples to understand better.

Example 1

```
import numpy as np
a = np.array([1,2,3])
print a
```

The output is as follows

```
[1, 2, 3]
```

3.3.1.2 Methods In Numpy

The array Method. To create a one-dimensional NumPy array, we can simply pass a Python list to the array method-

- The arrange Method
- The zeros Method
- The ones Method
- The line space Method
- The eye Method
- The random Method
- Indexing with 1-D Arrays.

3.3.2 Random Module

Python random module is an in-built module of Python which is used to generate the random numbers. These are pseudo-random numbers means these are not truly random. This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.

The random module is a built-in module to generate the pseudo-random variables. It can be used perform some action randomly such as to get a random number, selecting a random elements from a list, shuffle elements randomly, etc.

3.3.2.1 Generate Random Floats

The random.random() method returns a random float number between 0.0 to 1.0. The function doesn't need any arguments.

Example

```
random()
>>> import random
>>> random.random()
0.645173684807533
```

3.3.2.2 Generate Random Integers

The random.randint() method returns a random integer between the specified integers.

Example

```
randint()
>>> import random
>>> random.randint(1, 100)
95
```


3.3.2.3 Generate Random Numbers within Range

The `random.randrange()` method returns a randomly selected element from the range created by the start, stop and step arguments. The value of start is 0 by default. Similarly, the value of step is 1 by default.

Example

```
>>> random.randrange(1, 10)
2
>>> random.randrange(1, 10, 2)
5
>>> random.randrange(0, 101, 10)
80
```

3.3.2.4 Select Random Elements

The `random.choice()` method returns a randomly selected element from a non-empty sequence. An empty sequence as argument raises an Index Error.

Example

```
>>> import random
>>> random.choice('computer')
't'
>>> random.choice([12,23,45,67,65,43])
45
>>> random.choice((12,23,45,67,65,43))
67
```

3.3.2.5 Shuffle Elements Randomly

The `random.shuffle()` method randomly reorders the elements in a list.

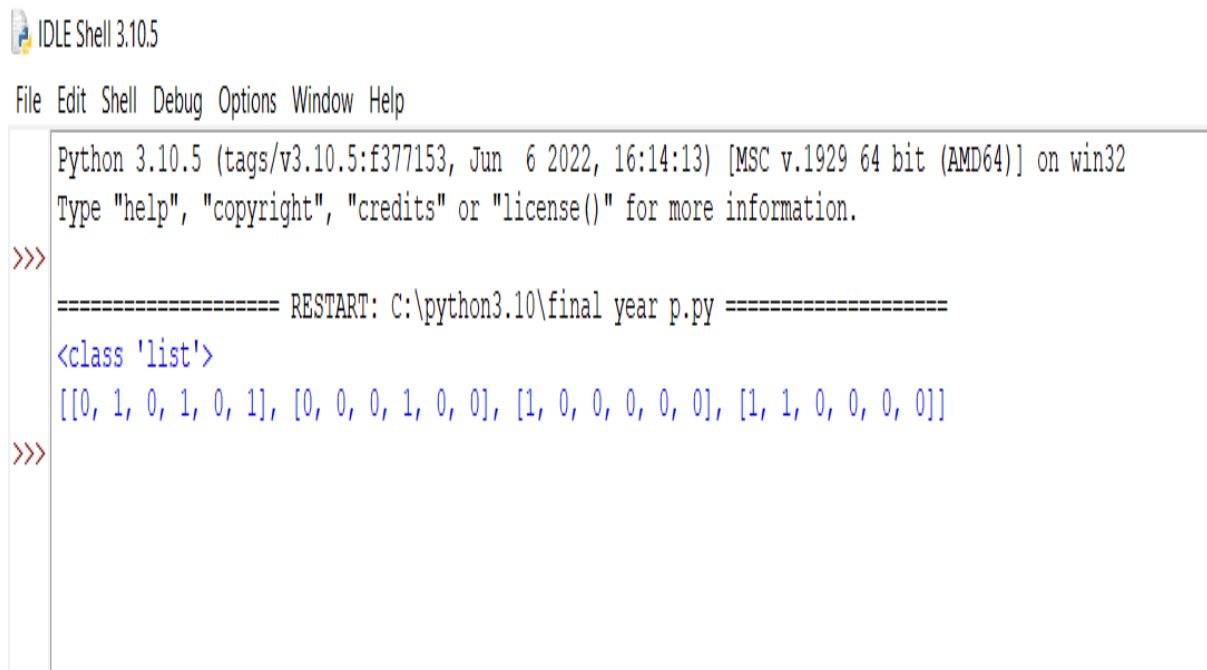
Example

```
>>> numbers=[12,23,45,67,65,43]
>>> random.shuffle(numbers)
>>> numbers
[23, 12, 43, 65, 67, 45]
>>> random.shuffle(numbers)
>>> numbers
[23, 43, 65, 45, 12, 67]
```

3.4 EXPECTED OUTCOME (PERFORMANCE METRICS WITH DETAILS)

3.4.1 Initial Population

```
import numpy as np # linear algebra
n=int(input("Enter number of generation ")) #initialize population
import random
best=-100000
populations = ([[random.randint(0,1) for x in range(6)] for i in range(4)])
print(type(populations))
parents=[ ]
new_populations = [ ]
print(populations)
```



```
IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\python3.10\final year p.py =====
<class 'list'>
[[0, 1, 0, 1, 0, 1], [0, 0, 0, 1, 0, 0], [1, 0, 0, 0, 0, 0], [1, 1, 0, 0, 0, 0]]
>>>
```

Fig.3.11. Initial Population

3.4.2 Fitness Score Calculation

```
#fitness score calculation
def fitness_score() :
    global populations,best
    fit_value = [ ]
    fit_score=[ ]
    for i in range(4) :
        chromosome_value=0
        for j in range(5,0,-1) :
            chromosome_value += populations[i][j]*(2**(5-j))
        chromosome_value = -1*chromosome_value if populations[i][0]==1 else
        chromosome_value
    print(chromosome_value)
    fit_value.append(-(chromosome_value**2) + 5 )
    print(fit_value)
    fit_value, populations = zip(*sorted(zip(fit_value, populations) , reverse = True))
    best= fit_value[0]
    fitness_score()
```

```
>>>
===== RESTART: C:\python3.10\final year p.py =====
<class 'list'>
[[0, 1, 0, 1, 1, 1], [0, 1, 1, 0, 1, 0], [0, 1, 1, 1, 1, 1], [1, 1, 1, 0, 0, 0]]
23
26
31
-24
[-524, -671, -956, -571]
>>>
```

Fig.3.12. Fitness Score Calculation

3.4.3 Selection Of Fittest Parents

```
#selecting parents
def selectparent():
    global parents
    #global populations , parents
    parents=populations[0:2]
    print(type(parents))
    print(parents)
    selectparent()
```

```
>>>
===== RESTART: C:\python3.10\final year p.py =====
<class 'list'>
[[1, 0, 1, 0, 0, 0], [1, 0, 0, 0, 0, 0], [1, 1, 0, 0, 1, 1], [0, 1, 0, 0, 1, 1]]
-8
0
-19
19
[-59, 5, -356, -356]
<class 'tuple'>
([1, 0, 0, 0, 0, 0], [1, 0, 1, 0, 0, 0])
>>>
```

Fig.3.13. Selection Of Fittest Parents

3.4.4 Single-Point Crossover

```
#single-point crossover
```

```
def crossover() :
```

```
    global parents
```

```
    cross_point = random.randint(0,5)
```

```
    parents=parents + tuple([(parents[0][0:cross_point +1] +parents[1][cross_point+1:6]))
```

```
    parents =parents+ tuple([(parents[1][0:cross_point +1] +parents[0][cross_point+1:6]))
```

```
    print(parents)
```

```
    crossover()
```

```
=====  
===== RESTART: C:\python3.10\final year p.py =====  
<class 'list'>  
[[1, 1, 0, 0, 0, 1], [1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 0, 1], [1, 1, 0, 0, 0, 1]]  
-17  
-7  
9  
-17  
[-284, -44, -76, -284]  
<class 'tuple'>  
([1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 0, 1])  
([1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 0, 1], [1, 0, 0, 1, 1, 1], [0, 0, 1, 0, 0, 1])  
>>>
```

Fig.3.14. Single-Point Crossover

3.4.5 Mutation

```
def mutation() :  
    global populations, parents  
    mute = random.randint(0,49)  
    if mute == 20 :  
        x=random.randint(0,3)  
        y = random.randint(0,5)  
        parents[x][y] = 1-parents[x][y]  
        populations = parents  
    print(populations)  
    mutation()
```

```
>>> ===== RESTART: C:\python3.10\final year p.py =====  
<class 'list'>  
[[1, 1, 0, 0, 1, 1], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [1, 0, 1, 0, 0, 0]]  
-19  
16  
4  
-8  
[-356, -251, -11, -59]  
<class 'tuple'>  
([0, 0, 0, 1, 0, 0], [1, 0, 1, 0, 0, 0])  
([0, 0, 0, 1, 0, 0], [1, 0, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0], [1, 0, 0, 1, 0, 0])  
([0, 0, 0, 1, 0, 0], [1, 0, 1, 0, 0, 0], [0, 0, 1, 0, 0, 0], [1, 0, 0, 1, 0, 0])  
>>>
```

Fig.3.15. Mutation

3.4.6 Iteration Of Generation

```
for i in range(n) :  
    fitness_score()  
    selectparent()  
    crossover()  
    mutation()  
    print("fitness score :")  
    print(best)  
    print("Number is...")  
    print(populations[0])
```

```
>>>===== RESTART: C:\python3.10\final year p.py =====  
<class 'list'>  
[[0, 1, 0, 1, 1, 1], [1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1], [1, 1, 1, 0, 1, 1]]  
23  
-9  
-23  
-27  
[-524, -76, -524, -724]  
<class 'tuple'>  
([1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1])  
([1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1], [1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1])  
([1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1], [1, 0, 1, 0, 0, 1], [1, 1, 0, 1, 1, 1])  
-9  
-23  
-9  
-23  
[-76, -524, -76, -524]  
<class 'tuple'>  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
-9  
-9  
-9  
-9  
[-76, -76, -76, -76]  
<class 'tuple'>  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
([1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1], [1, 0, 1, 0, 0, 1])  
fitness score :  
-76  
Number is...  
[1, 0, 1, 0, 0, 1]  
>>>|
```

Fig.3.16. Iteration of generation

CONCLUSION

After this much analysis and implementation of the algorithm, now we are in a position to make some conclusion. The purpose of this project was to make use of genetic algorithm to find the most optimal solution of polynomial. Based on the analysis, we can conclude that taking help of some random values of solution of polynomial, new values can be generated which will give most optimal solution for polynomial than previously selected values.

REFERENCES

- [1] Anup Kulkarni <https://images.app.goo.gl/nFqTi4yPewHHLpw37>
- [2] R.K. Bhattacharjya <https://www.iitg.ac.in/rkbc/CE602/CE602/Genetic%20Algorithms.pdf>
- [3] Atul Kuamr <https://www.geeksforgeeks.org/genetic-algorithms/>
- [4] IEEE <https://ieeexplore.ieee.org/document/9002372>
- [5] Wikipedia https://en.wikipedia.org/wiki/Genetic_algorithm
- [6] Tutorials Point https://www.tutorialspoint.com/genetic_algorithms/index.htm