Project Title

Report submitted for the partial fulfillment of the requirements for the degree of Bachelor of Technology in

Information Technology

Submitted by

AFRIN KHATUN -- (11700214005)

SHIKHA SINGH – (11700214063)

Under the Guidance of

Dr. Dipankar Majumdar

(Associate Professor, Of Information Technology)

RCC Institute of Information Technology

Canal South Road, Beliaghata, Kolkata – 700 015

[Affiliated to West Bengal University of Technology]

Acknowledgement

We would like to express our sincere gratitude to Dr. Dipankar Majumdar of the department of Information Technology, whose role as project guide was invaluable for the project. We are extremely thankful for the keen interest he / she took in advising us, for the books and reference materials provided for the moral support extended to us.

Last but not the least we convey our gratitude to all the teachers for providing us the technical skill that will always remain as our asset and to all non-teaching staff for the gracious hospitality they offered us.

Place: RCCIIT, Kolkata

Date:15.05.2018

……

AFRIN KHATUN

……

SHIKHA SINGH

Department of Information Technology
RCCIIT, Beliaghata,
Kolkata – 700 015,
West Bengal, India

Approval

This is to certify that the project report entitled "INTELLIGENT SCHEDULING SYSTEM USING GENETIC ALGORITHM" prepared under my supervision by AFRIN KHATUN (Roll no. -11700214005) and SHIKHA SINGH (Roll no - 11700214063) be accepted in partial fulfillment for the degree of Bachelor of Technology in Information Technology.

It is to be understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn thereof, but approves the report only for the purpose for which it has been submitted.

Dr. Abhijit Das	Dr. Dipankar Majumdar (Associate Professor)
Name & Designation of the HOD	Name & Designation of Internal Guide

INTRODUCTION

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. **It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.**

Optimization is the process of **making something better**. In any process, we have a set of inputs and a set of outputs as shown in the following figure.

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.

It is frequently used to solve optimization problems, in research, and in machine learning.Nature has always been a great source of inspiration to all mankind. Genetic Algorithms (GAs) are search based algorithms based on the concepts of **natural selection and genetics.**

GAs were developed by John Holland and his students and colleagues at the University of Michigan, most notably David E. Goldberg and has since been tried on various optimization problems with a high degree of success.

In GAs, we have a **pool or a population of possible solutions** to the given problem. These **solutions** then undergo **cross-over and mutation** (like in natural genetics), producing new children, and the process is **repeated over various generations**. Each individual (or candidate solution) is assigned a **fitness value** (based on its objective function value) and the fitter individuals are given a **higher chance** to mate and yield more **"fitter" individuals.** This is in line with the **Darwinian Theory of "Survival of the Fittest".**

In this way we keep "evolving" **better** individuals or solutions over generations, till we reach a stopping criterion.

Genetic Algorithms are sufficiently randomized in nature, but they perform much better than random local search (in which we just try various random solutions, keeping track of the best so far), as they exploit historical information as well.

PROBLEM DEFINITION

PROBLEM-1

- ‘n’ aircrafts need to be scheduled.
- These ‘n’ aircrafts have different arrivaland departure time
- Only one aircrafts can carry at a time (only one runway is available)
- Aircrafts need to make prior bookings with airport authority.
- The airport authority have fixed the charges per unit time. They also compensate the aircrafts agencies in case of delay.This amount is also fixed per unit time.
- Schedule the ‘n’ aircrafts in such a way that the airport authorities makes max income.

PROBLEM-2

- Solve the above problem with the same constraints as stated above such that only two aircrafts can carry out their activity at a time (only two runways are available).

Domain-: C programming language

LITERATURE SURVEY

Previously we have worked on the problem using genetic algorithm where we had schedule n jobs. There were two machines A and B. Each job had different processing time for two different machine.We had to schedule the jobs in such a way on the two machines that they be executed in the least possible time.The processors were independent of each other.All the jobs were executed in the least possible time.

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of **Genetics and Natural Selection**. **It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve.**

We start with an initial population (which may be generated at random or seeded by other heuristics), select parents from this population for mating. Apply crossover and mutation operators on the parents to generate new off-springs. And finally these off-springs replace the existing individuals in the population and the process repeats. In this way genetic algorithms actually try to mimic the human evolution to some extent. One of the most important decisions to make while implementing a genetic algorithm is deciding the representation that we will use to represent our solutions. It has been observed that improper representation can lead to poor performance of the GA.

Therefore, choosing a proper representation, having a proper definition of the mappings between the phenotype and genotype spaces is essential for the success of a GA.

In this section, we present some of the most commonly used representations for genetic algorithms. However, representation is highly problem specific and the reader might find that another representation or a mix of the representations mentioned here might suit his/her problem better.

PLANNING

Structure templates used:

1.To represent the chromosomes

```
typedef struct chrom
{
int bit[MAX];
int fit;
}chrom;
```

This structure template called chrom contains bit which represent the chromosome and fit represent their corresponding fitness value.

	bit[0]	bit[1]	bit[2]	bit[3]	bit[4]	……..	bit [n-1]
row 1	0	1	0	0	1	……..	0
row 2	1	1	1	1	0	……..	0
row 3	0	0	0	1	1	……..	1
row 4	1	1	1	0	0	……..	0

CHROMOSOME REPRESENTATION

We take an arrray of structure of chrom type of size 4 in order to save 4 chromosomes for the population. Each population contains 4 chromosomes.
We generate the population of 4 chromosomes randomly

- each row is a chromosomes
- each row(chromosome) consist of 'n' number of genes.

i.e, here n means the number of genes = number of aircrafts to be scheduled and a fitness value is calculated for each chromosomes.The chromosomes are generated randomly.
We make 2 population set , where a current set has the initial four chromosomes and the next set contains chromosomes produced by the first set and so on until we finish the number of iterations.

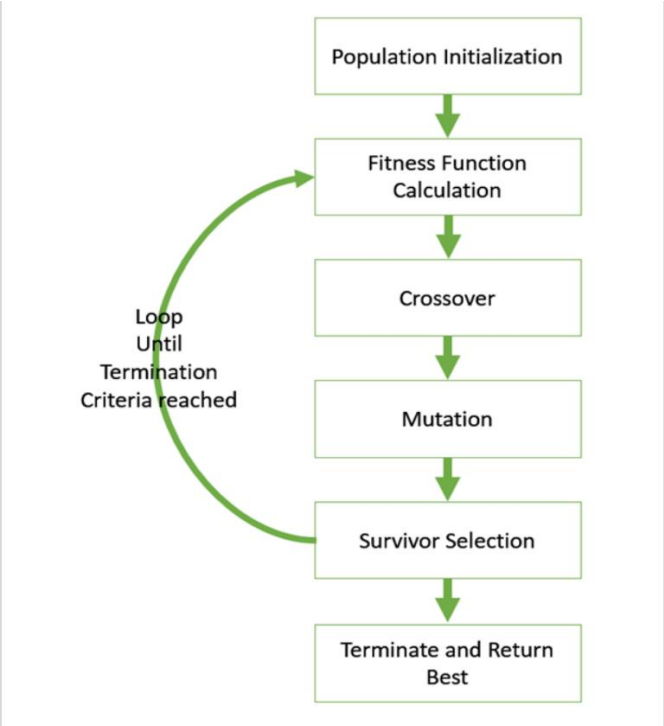
We have use chrom final:-
final structure of type chrom saves the best chromosomes foud during the full iteration .

2.To represent the aircrafts.

```
typedef struct Aircraft
        {
                int arrival_time;
                int departure_time;
                int cost;
                int time;
        }aircraft;
```

We take an arrray of structure of aircraft type of in order to save the information of all the aircrafts.i.e,their arrival times, departure times, the time that they will require to complete their jobs and the cost incurred by them.We make an array of structure of type aircrafts of size n. Here ‘n’ is the number of aircrafts to be scheduled.

Working of Genetic Algorithm:



Algorithm to solve the problem 1 and 2

integer number,payback,earn are declared as global variables
aircrafts[max] is globally declare

Step 1: Begin
Step 2: Declaration int num,i,j,k,maxprofit=0;
Step 3: Print (welcome to scheduling)
Print (enter the number of aircrafts)
Read number
Print (enter the amount airport authority charges per unit time)
Read earn
Print (enter the amount agency pays back per unit time)
Read Payback
Step 4: create function generate(aeroplane)
Step 5: sort (aeroplane)
sortingwithin(aeroplane)
Step 6:Print(Please enter the no. of iterations)
Step 7:Read num
Step 8:chrom popcurrent[4]
chrom popnext[4]
Step 9: for loop(i=0 to num)
9.1: Print iteration-->i
9.2:for(j=0 to j<4)
popnext[j]=popcurrent[j]
9.3:end of for(j)

pickchroms(popnext)
check(popnext)
crossover(popnext)
check(popnext)
mutation(popnext)
check(popnext)

9.4: for(j=0 to j<4)
popcurrent[j]=popnext[j]

9.5:endof for(j)

Step 10:end of for(i)
Step 11:Print popcurrent
Step 12:Print(the best chromosome found so far is)
Step 13:Print final
Step 14:Print(0 means that job is rejected,1 means the job is chosen)
Step 15:printingthebestsolution(final)
Step16:Stop

PROBLEM DESIGN

FUNCTIONS USED

1.Picking the best chromosomes

Gentic Algorithm is based on **"survival of the fittest"**.So we choose the first two best chromosomes from the old population according to the fitness value(whoever has the highst fitness function is chosen)
We do so by sorting the 4 chromosomes accrording to their fitness vaues.(in the descending order).

Algorithm for picking the best chromosomes

pickchroms(chrom popnext[4])
Step 1:Begin
Step 2:Declaration- int i,j,k;
Step 3:make a structute called temp of type chrom
Step 4:for loop(i=0 to i<3)
    4.1:for loop(j=0 to j<3)
        if(popnext[j].fit<popnext[j+1].fit)
            then swap(popnext[j],popnext[j+1])with the help of temp
    4.2:end of for (j)
Step 5:end of for if
Step 6:for(i=0 to i<4)//to print the chromosomes and their fitness values after sorting
    6.1:for(k=0 to k<(no of aircrafts))
        print popnext[i].bit[k]
    6.2:end of for k
    6.3:print popnext[i].fit
Step 7:end of for (i)
Step 8:Stop

2.Checking the population for a chromosomes with a highest fitness value than that of final chromosome.

If we find a better chromosomes than that present in final,then we will save the better chromosomes in final.

Algorithm for searching the best chromosome and to save it in final

check(chrom popnext[4])
step1: Begin
step2:Declaration int i;
step 3: for loop(j=0 to j<4)
    3.1 if(popnext[j].fit> final.fit)
        final =popnext[j]
step 4: end of for (j)
step 5:stop

3.Cross Over to generate the children

We have picked the best two chromosomes .We create the children by crossing over these two best chromosomes to create two children.The new population
is thus going to contain the two best chromosomes and their children.Single point crossover is conduct over here.

Algorithm for cross-over

crossover(chrom popnext[4])
Step 1:Begin
Step 2:Declaration- int i,random;
Step 3:random=rand()
Step 4:random=random%(no of aircrafts)//choosing cross-over point
Step 5:for loop(i=0 t0 i<random)

5.1:copy the bit[i] of 1st chromosome to bit[i] of 3rd chromosome
5.2:copy the bit[i] of 2nd chromosome to bit[i] of 4th chromosome
Step 6:end of for
Step 7:for loop(i=random to i<no of aircrafts)
7.1:copy the bit[i] of 2nd chromosome to bit[i] of 3rd chromosome
7.2:copy the bit[i] of 1st chromosome to bit[i] of 4th chromosome
Step 8:end of for
Step 9:for loop(i=0 t0 i<4)
9.1:calculate the fitness of each chromosome
Step 10:end of for
Step 11:Print the new population after doing crossover
Step 12:Stop

4.Mutation with a high probability

Mutation occurs in one chromosome in the set by inverting any one bit in the chromosomes randomly.

Algorithm for mutation in problem 1

mutation(chrom popnext[4])
Step 1:Begin
Step 2:Declaration- int random,col,value;
Step 3:random=rand()
Step 4:random=rand()%50
Step 5:if(random>25)
5.1:col=rand()%(no of aircrafts)//choosing the gene
5.2:row=rand()%4//choosing the chromosome
5.3:if the chosen gene of the chosen chromosome i.e popcurrent[row].bit[col] == 0
make it 1
5.4:if the chosen gene of the chosen chromosome i.e popcurrent[row].bit[col] == 1
make it 0
Step 6:end of if i.e step 5
Step 7:calculate the fitness function of the mutated chromosome
Step 8:print the mutated chromosome and its fitness value
Step 9:Stop

Algorithm for mutation for mutation in problem 2

mutation(chrom popnext[4])
Step 1:Begin
Step 2:Declaration- int random,col,value;
Step 3:random=rand()
Step 4:random=rand()%50
Step 5:if(random>25)
5.1:col=rand()%(no of aircrafts)//choosing the gene
5.2:row=rand()%4//choosing the chromosome
5.3:if the chosen gene of the chosen chromosome i.e popcurrent[row].bit[col] == 0
temporaryone = popnext[row]
temporarytwo = popnext[row]
temporaryone.bit[col]=1
calculate temporaryone.fit
temporarytwo.bit[col]=2
calculate temporarytwo.fit
5.3.1:if(temporaryone.fit > temporarytwo.fit)
popnext[row]=temporaryone
5.3.2:else
popnext[row]=temporarytwo
5.4:end of if

5.5:if the chosen gene of the chosen chromosome i.e popcurrent[row].bit[col] == 1
temporaryone = popnext[row]
temporarytwo = popnext[row]
temporaryone.bit[col]=0
calculate temporaryone.fit
temporarytwo.bit[col]=2
calculate temporarytwo.fit
5.5.1:if(temporaryone.fit > temporarytwo.fit)
popnext[row]=temporaryone
5.5.2:else
popnext[row]=temporarytwo
5.6:end of if
5.7:if the chosen gene of the chosen chromosome i.e popcurrent[row].bit[col] == 2
temporaryone = popnext[row]
temporarytwo = popnext[row]
temporaryone.bit[col]=0
calculate temporaryone.fit
temporarytwo.bit[col]=1
calculate temporarytwo.fit
5.7.1:if(temporaryone.fit > temporarytwo.fit)
popnext[row]=temporaryone
5.7.2:else
popnext[row]=temporarytwo
5.8:end of if
Step 6:end ofif
Step 7:calculate the fitness function of the mutated chromosome
Step 8:print the mutated chromosome and its fitness value
Step 9:Stop

5.Loop Termination

The number of iteration are entered by the user. After every iteration we get a new set of chromosomes and the steps is carried out in this specific order -->1,2,3,2,4,2

6.Sorting the aircrafts according to arrival times
We are going to sort the aircrafts according to their arrival times (in the ascending order)We do so to maintain the order in which the aircraft are coming.

Algorithm for sorting the aircraft according to their arrival times
sort(aircraft aeroplane[no of aircrafts])
Step 1:Begin
Step 2:Declaration- int i,j;
Step 3:make a structute called temp of type aircraft
Step 4:for loop(i=0 to i<3)
4.1:for loop(j=0 to j<3)
if(aeroplane[j].arrival_time>aeroplane[j+1].arrival_time)
then swap(aeroplane[j],aeroplane[j+1])with the help of temp
4.2:end of for (j)
Step 5:print aeroplane in the sorted order
Step 6:Stop

7.Sorting the aircrafts with the same arrivals times according to their departure times.

We have already sorted the aircrafts according to their arrivals times.Now we will sort the ones with the same arrivals time according

to their departure time(in the ascending order)

Algorithm for sorting the aircraft with the same arrival times according to their departure times

```
sortingwithin(aircraft aeroplane[no of aircrafts])
Step 1:Begin
Step 2:Declaration- int p,c,k,i,j;
Step 3:Initialise c to 0
Step 4:for(p=0 to p<(number of aircrafts))
        4.1: if(aeroplane[p].arrival_time == aeroplane[p+1].arrival_time)
                then c++
                     p++
                4.1.1: check if(aeroplane[p].arrival_time == aeroplane[p+1].arrival_time)
                              if true then goto Step 4.1
        4.2: end of if
        4.3: if(c!=0)
                4.3.1: k=p-c
                4.3.2: for(i=k to i<=p)
                  4.3.2.1: for(j=k to j<=p)
                                  if(aeroplane[i].departure_time < aeroplane[j].departure_time)
                                  then swap(aeroplane[j],aeroplane[i])with the help of temp
                        4.3.2.2:end of for(j)
                4.3.3:end of for(i)
        4.4:end of if i.e, Step 4.3
        4.5:c=0
Step 5:end of for(p)
Step 6:Stop
```

8.Printing the solution

This function is used to convert the genotype into the pheno-type.So this is used to decode the solution chromosomes to represent the actual real word solution space.

9.Generating the population of chromosomes

Algorithm for generating the population containing four chromosomes in problem1

```
evpop(chrom popcurrent[4])
Step 1:Begin
Step 2:Declaration- int i,j,value,k,random;
Step 3:for(j=0 to j<4)
        3.1:for(i=0 to i<(number of aircrafts))
                        random=rand()
                        random=random%2
                        popcurrent[j].bit[i]=random
        3.2end of for(i)
        3.3 calculation the fitness function of the chromosome popcurrent[j]
Step 4:end of for(j)
Step 5:for(i=0 to i<4)//to print the chromosomes and their fitness values
        5.1:for(k=0 to k<(no of aircrafts))
                print popcurrent[i].bit[k]
        5.2:end of for k
        5.3:print popcurrent[i].fit
Step 6:end of for (i)
Step 7:Stop
```

Algorithm for generating the population containing four chromosomes in problem2

evpop(chrom popcurrent[4])
Step 1:Begin
Step 2:Declaration- int i,j,value,k,random;
Step 3:for(j=0 to j<4)
3.1:for(i=0 to i<(number of aircrafts))
random=rand()
random=random%2
popcurrent[j].bit[i]=random
3.2end of for(i)
3.3 calculation the fitness function of the chromosome popcurrent[j]
Step 4:end of for(j)
Step 5:for(i=0 to i<4)//to print the chromosomes and their fitness values
5.1:for(k=0 to k<(no of aircrafts))
print popcurrent[i].bit[k]
5.2:end of for k
5.3:print popcurrent[i].fit
Step 6:end of for (i)
Step 7:Stop

10.Generation of aeroplane which is an array of structures

Algorithm for generating the array of structure called aeroplane of type aircraft containing the information of all the aircrafts

generate(aircraft aeroplane[number of aircrafts])
Step 1:Begin
Step 2:Declaration- int i;
Step 3:for(i=0 to i<(number of aircrafts))
3.1:Read aeroplane[i].arrival_time
3.2:Read aeroplane[i].departure_time
3.3:Calculate aeroplane[i].cost=((aeroplane[i].departure_time-aeroplane[i].arrival_time)*earn)
3.4:aeroplane[i].time=aeroplane[i].departure_time-aeroplane[i].arrival_time
Step 4:end of for(i)
Step 5:Stop

PROBLEM SPECIFIC PART

CHROMOSOME REPRESENTATION OF PROBLEM 1

0 means that the job is rejeted 0.
1 means that the job will be executed on runway 1.
So we are using 0 and 1 only.So each gene of the chromosome has the value 0 or 1. For 'n' aircrafts each chromosome contains n genes.

FITNESS FUNCTION OF PROBLEM 1

We maintain the variable c,z, and m.c keeps track of the current time.
z contain the total amount that can be earned if all the jobs were executed on time without any delay.
m contains the total amount that need to be paid back to all aircrafts for causing them delay.
Total profit =>z-m;

Algorithm for calculating fitness function of problem 1

```
x(chrom popcurrent)
Step 1:Begin
Step 2:Declaration- int i,k,c,m,z;
Step 3:Initialise z and m to 0
Step 4:for loop(i=random to i<no of aircrafts)
        4.1:if popcurrent.bit[i]==1
            then k=i;
                 c=aeroplane[k].departure_time;
                 z=aeroplane[k].cost;
                 goto step 6
Step 5:end of for (i)
Step 6:for loop(k+1 to number)
        6.1: if popcurrent.bit[i]==1
            then
            z=z+aeroplane[i].cost;
            if(c>aeroplane[i].arrival_time){
                m=m+((c-aeroplane[i].arrival_time)*pay_back)
            }
            else{
                c=aeroplane[i].departure_time;
            }
        6.2:End of if i.e Step 6.1
Step 7:End of for(i)
Step 8:k=z-m
Step 9:Stop
```

CHROMOSOME REPRESENTATION OF PROBLEM 2

0 means that the job is rejeted 0.
1 means that the job will be executed on runway 1.
2 means that the job will be executed on runway 2.
So we are using 0,1 and 2 only.So each gene of the chromosome has the value 0 or 1. For 'n' aircrafts each chromosome contains n genes.

FITNESS FUNCTION OF PROBLEM 2

We maintain 4 variables d,c,z and m. c keeps track of the current time on runway one.d keeps track of the current time on runway two.
z cointains the total amount that can be earned if all the jobs were executed on time(on runway 1 as well as on runway two).m contains the total amount that needs to be paid back to all the aircrafts for causing them delay (on runway 1 as well as on runway 2).

Algorithm for calculating fitness function of problem 2

```
x(chrom popcurrent)
Step 1:Begin
Step 2:Declaration- int c,d,z,k,m,i;
Step 3:Initialise c,d,z,k and m to 0
Step 4:for(i=0 to i<(number of aircrafts))
        4.1:check if((popcurrent.bit[i]==1)&&(i!=0)&&(c==0))
                then c=aeroplane[i].departure_time
                and z=z+aeroplane[i].cost
        4.2:check else if((popcurrent.bit[i]==2)&&(i!=0)&&(d==0))
                then d=aeroplane[i].departure_time
```

and z=z+aeroplane[i].cost

4.3:check else if((popcurrent.bit[i]==1)&&(i==0)&&(c==0))
then z=z+aeroplane[i].cost
and c=aeroplane[i].departure_time

4.4:check else if((popcurrent.bit[i]==2)&&(i==0)&&(d==0))
then z=z+aeroplane[i].cost
and d=aeroplane[i].departure_time

4.5:check else if((popcurrent.bit[i]==1)&&(i!=0)&&(c!=0))
then z=z+aeroplane[i].cost

4.5.1: if(c>aeroplane[i].arrival_time)
m=(m+((c-aeroplane[i].arrival_time)*pay_back))
c=c+aeroplane[i].time

4.5.2: else
c=aeroplane[i].departure_time

4.6:end of Step 4.5

4.7:check else if((popcurrent.bit[i]==2)&&(i!=0)&&(d!=0))
z=z+aeroplane[i].cost;

4.7.1: if(d > aeroplane[i].arrival_time)
m=(m+((d - aeroplane[i].arrival_time)*pay_back));
d=d+aeroplane[i].time;

4.7.2: else
d=aeroplane[i].departure_time;

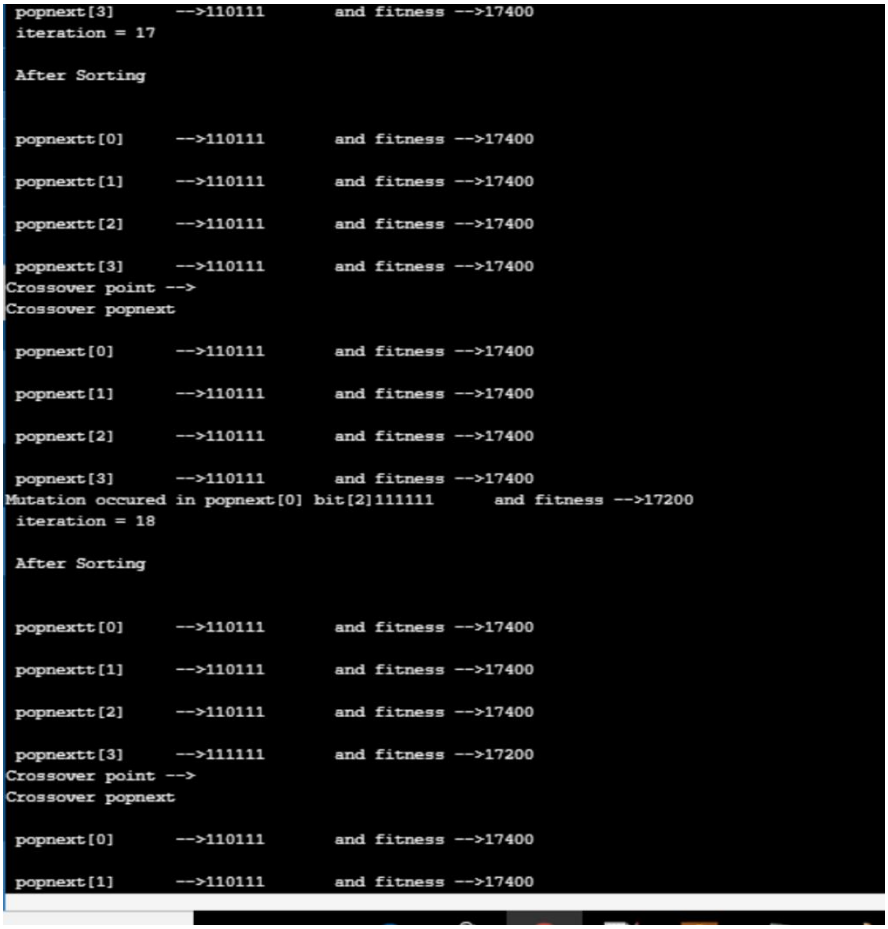
4.8:end of Step 4.7

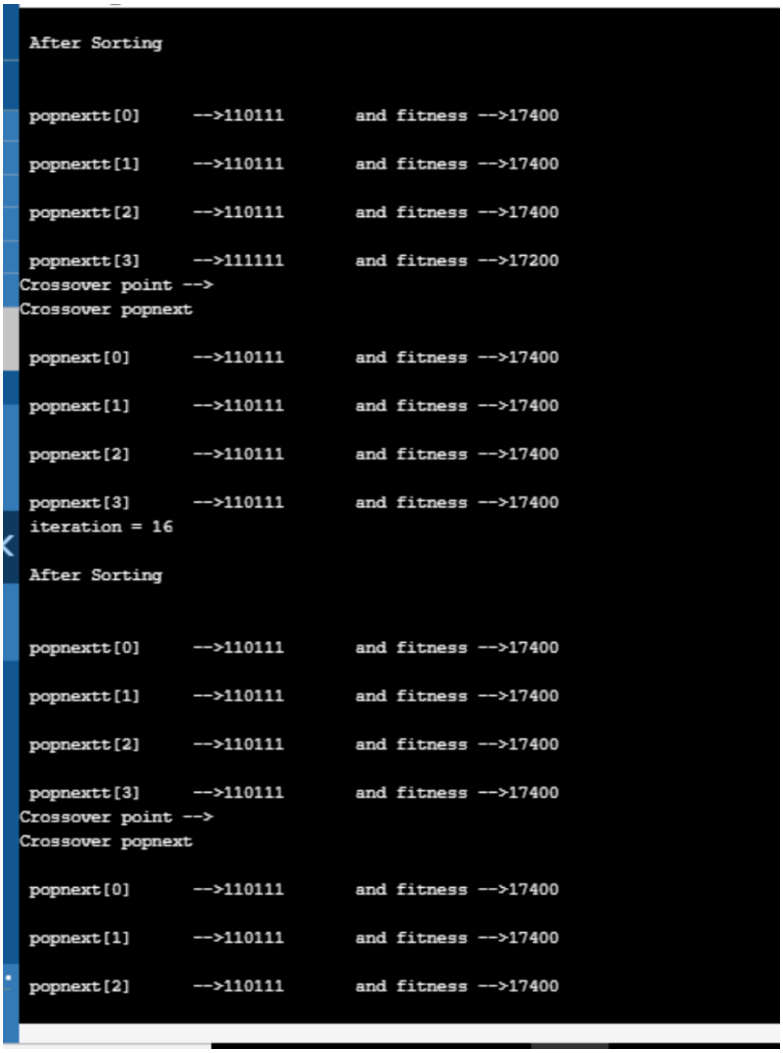
4.9:else z=z+0

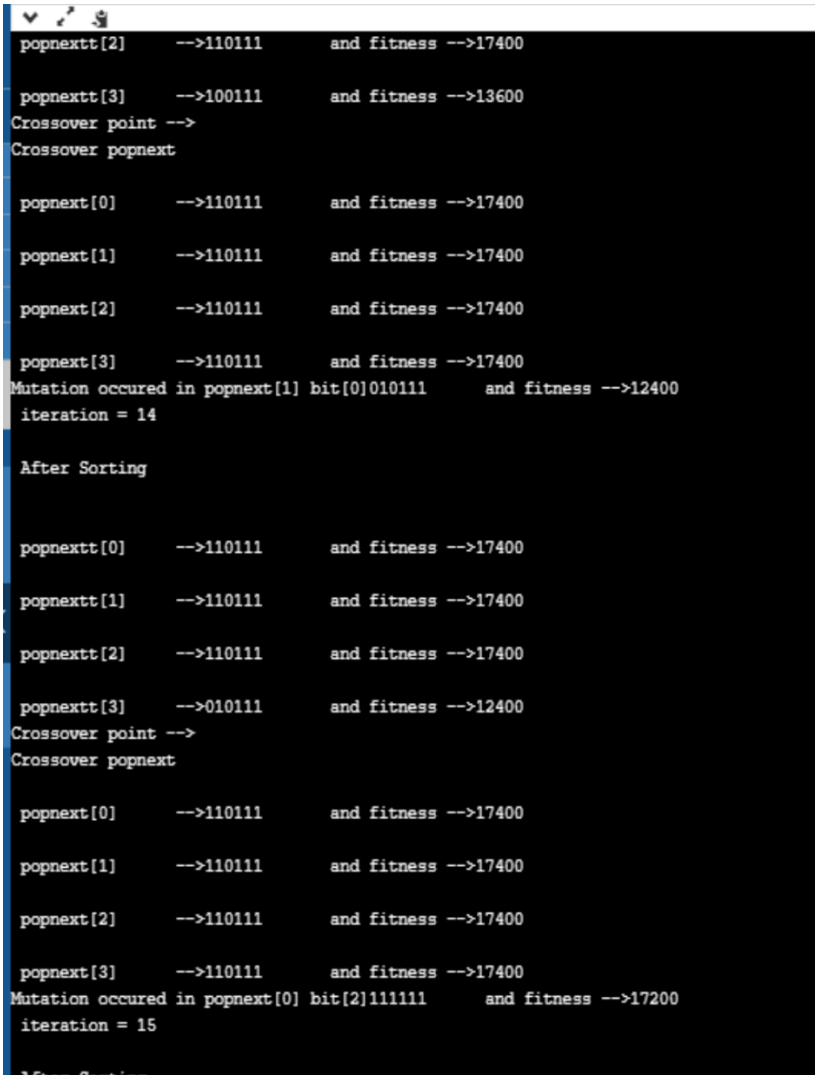
Step 5:k=z-m and return k

Step 6:Stop

RESULT


popnext[3] -->110111 and fitness -->17400
iteration = 17
After Sorting
popnextt[0] -->110111 and fitness -->17400
popnextt[1] -->110111 and fitness -->17400
popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->110111 and fitness -->17400
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400
popnext[2] -->110111 and fitness -->17400
popnext[3] -->110111 and fitness -->17400
Mutation occured in popnext[0] bit[2]111111 and fitness -->17200
iteration = 18
After Sorting
popnextt[0] -->110111 and fitness -->17400
popnextt[1] -->110111 and fitness -->17400
popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->111111 and fitness -->17200
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400


After Sorting
popnextt[0] -->110111 and fitness -->17400
popnextt[1] -->110111 and fitness -->17400
popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->111111 and fitness -->17200
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400
popnext[2] -->110111 and fitness -->17400
popnext[3] -->110111 and fitness -->17400
iteration = 16
After Sorting
popnextt[0] -->110111 and fitness -->17400
popnextt[1] -->110111 and fitness -->17400
popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->110111 and fitness -->17400
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400
popnext[2] -->110111 and fitness -->17400


popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->100111 and fitness -->13600
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400
popnext[2] -->110111 and fitness -->17400
popnext[3] -->110111 and fitness -->17400
Mutation occured in popnext[1] bit[0]010111 and fitness -->12400
iteration = 14
After Sorting
popnextt[0] -->110111 and fitness -->17400
popnextt[1] -->110111 and fitness -->17400
popnextt[2] -->110111 and fitness -->17400
popnextt[3] -->010111 and fitness -->12400
Crossover point -->
Crossover popnext
popnext[0] -->110111 and fitness -->17400
popnext[1] -->110111 and fitness -->17400
popnext[2] -->110111 and fitness -->17400
popnext[3] -->110111 and fitness -->17400
Mutation occured in popnext[0] bit[2]111111 and fitness -->17200
iteration = 15

```
popnext[0]      -->110111       and fitness -->17400

popnext[1]      -->110111       and fitness -->17400

popnext[2]      -->110111       and fitness -->17400

popnext[3]      -->110111       and fitness -->17400
Mutation occured in popnext[1] bit[2]111111      and fitness -->17200
 iteration = 12

 After Sorting


popnextt[0]     -->110111       and fitness -->17400

popnextt[1]     -->110111       and fitness -->17400

popnextt[2]     -->110111       and fitness -->17400

popnextt[3]     -->111111       and fitness -->17200
Crossover point -->
Crossover popnext

popnext[0]      -->110111       and fitness -->17400

popnext[1]      -->110111       and fitness -->17400

popnext[2]      -->110111       and fitness -->17400

popnext[3]      -->110111       and fitness -->17400
Mutation occured in popnext[1] bit[1]100111      and fitness -->13600
 iteration = 13

 After Sorting


popnextt[0]     -->110111       and fitness -->17400

popnextt[1]     -->110111       and fitness -->17400

popnextt[2]     -->110111       and fitness -->17400
```

```
 Enter the arrival time for aeroplane 4
2

 Enter the departure time for aeroplane 4
9

 Enter the information for aeroplane 5

 Enter the arrival time for aeroplane 5
5

 Enter the departure time for aeroplane 5
9

 Enter the information for aeroplane 6

 Enter the arrival time for aeroplane 6
4

 Enter the departure time for aeroplane 6
7

 Sorting done based on arrival times of the aircraft: aeroplane[0].arrival_time = 1

 Sorting done based on arrival times of the aircraft: aeroplane[1].arrival_time = 1

 Sorting done based on arrival times of the aircraft: aeroplane[2].arrival_time = 2

 Sorting done based on arrival times of the aircraft: aeroplane[3].arrival_time = 2

 Sorting done based on arrival times of the aircraft: aeroplane[4].arrival_time = 4

 Sorting done based on arrival times of the aircraft: aeroplane[5].arrival_time = 5

 aeroplane[0] --> arrival_time = 1 --> departure_time = 6 --> cost = 4500

 aeroplane[1] --> arrival_time = 1 --> departure_time = 8 --> cost = 6300

 aeroplane[2] --> arrival_time = 2 --> departure_time = 4 --> cost = 1800

 aeroplane[3] --> arrival_time = 2 --> departure_time = 9 --> cost = 6300
```

CONCLUSION

Genetic Algorithms have the ability to deliver a “good-enough” solution “fast-enough”. This makes genetic algorithms attractive for use in solving optimization problems. It does not require any derivative information (which may not be available for many real-world problems).It 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. It provides a list of “good” solutions and not just a single solution. We 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.

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.

REFERENCES

1. Eiben, A. E. et al (1994). "Genetic algorithms with multi-parent recombination". PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: 78–87. ISBN 3-540-58484-6.

2. Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430 [1]

3. www.tutorialspoint.com/genetic_algorithm/index.html