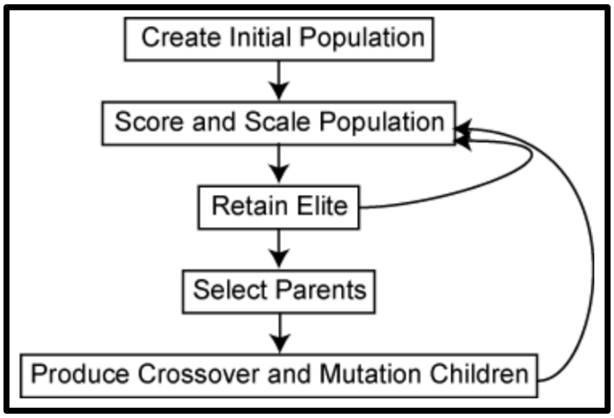
1. Problem Statement

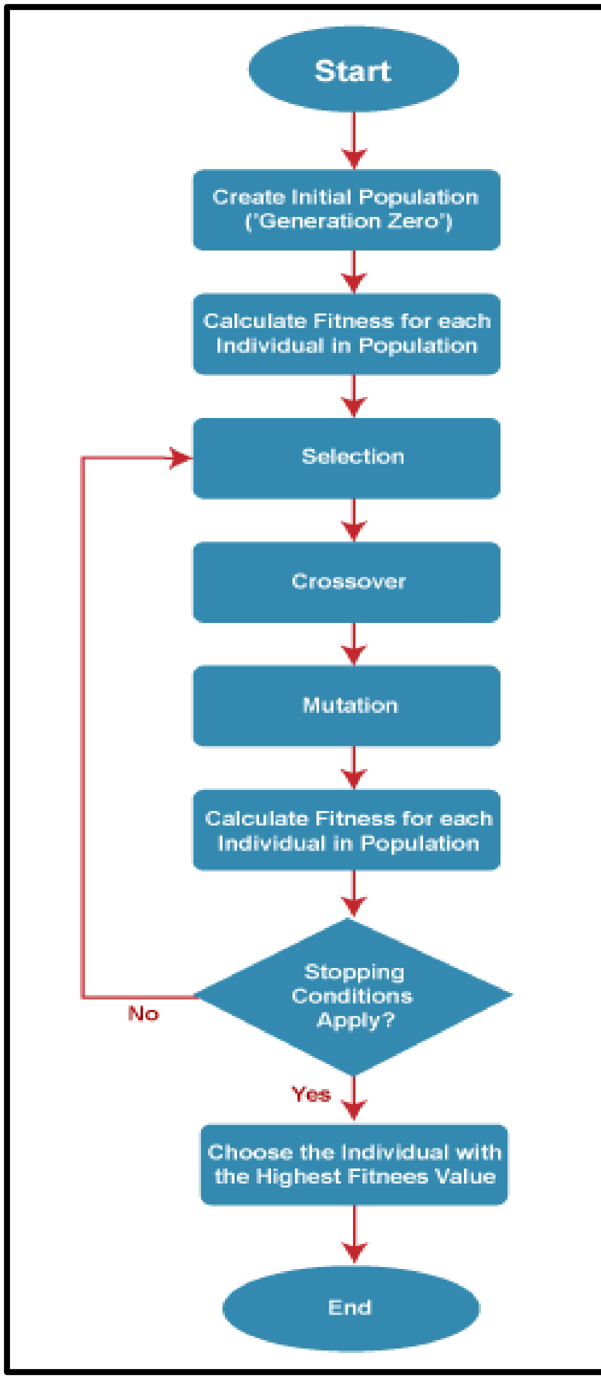
Use a binary-coded GA to minimize the function $f(x_1, x_2) = x_1 + x_2 - 2x_1^2 - x_2^2 + x_1x_2$, in the range of $0.0 \le x_1, x_2 \le 0.5$. Using a random population of size N=6 and assume 5 bits for each variable.

2. Parameters used

Number of Variables	2
Number of Bits for each variable	5
Total Number of Bits	10
x_{min}, x_{max}	0, 0.5
Minimization Problem converted to maximization using	$F(x) = -f(x)$
Population Size (N)	6
Maximum Number of Generation	1,00,000
Selection technique	Tournament and Elitism
Elitism Selection size	0 (no elite is retained)
Tournament Selection size	3
Crossover Probability (Pc)	0.9
Mutation Probability (Pm)	0.1

3. Procedure for Binary Coded Genetic Algorithm





a. Initialize a population of size N=6 with 6 strings of length 10 bits at random (5 bits for each variable).

b. Then decoded values all strings with first five bit for 1st variable (x_1) and next five bit for 2nd variable(x_2).

c. The real values of x1 and x2 are then computed using the formula below.

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{\ell_i} - 1} DV(\mathbf{s}^i),$$

d. The fitness value is then determined. The fitness function F(x) = -f is used to turn the minimization problem into a maximization problem.

e. Then, by tournament selection of size 3, reproduction is carried out, and winners are obtained.

f. Then, using the Elitism approach, the best answers can be maintained the same (for our problem zero elite is retained).

g. The Single Point Crossover technique is then used with a probability of pc = 0.9 for the remaining four solutions.

h. The population is then subjected to mutations with a mutation probability of pm = 0.1.

i. After then, the procedure is repeated for 100000 generations.

4. Results

Initial population :
1. 1010011101
2. 1001100000
3. 0101000011
4. 1010100000
5. 0001100100
6. 0010100011

Final Population after 100000 generation :
1. 0000000000
2. 0000000000
3. 0000000000
4. 0000000000
5. 0000000000
6. 0000000000

The final value of all population x_1, x_2, $f(x_1,x_2)$ and fitness values are

Population #	x_1	x_2	$f(x_1, x_2)$	Fitness value
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0

The minimum of given problem lies at x1 = 0, x2 = 0, and minimum function value is $f(x_1,x_2) = 0$.

There is another minimum solution of this problem which lies at x1 = 0.5, x2 = 0, and $f(x_1,x_2) = 0$.

CODE is attached below.

```
#include<bits/stdc++.h>

using namespace std;


double f(double , double );


int main(){
    int n = 6, l1 = 5, l = 10, no_of_variable = 2;
    double xmin = 0, xmax = 0.5;
    int e = 0; // no of element for elitsm (keep even number if n is even, and odd if n is odd)
    int toursize = 3; // no. of element select for tournament
    double pc = 0.9, pm = 0.1; // crossover and mutation probability
    int max_generation = 100000;


    vector <vector <int> > POPULATION( n+1, vector<int> (l+1));
    vector <vector <int> > DECODE_VALUE( n+1, vector<int> (no_of_variable+1));
    vector <vector <double> > X( n+1, vector<double> (no_of_variable+1));
    vector <vector <double> > FUNCTION_VALUE( n+1, vector<double> (2));
    vector <vector <double> > FITNESS( n+1, vector<double> (2));
    vector <vector <int> > POOL( n+1, vector<int> (l+1));
    vector <vector <int> > CROSSOVER( n+1, vector<int> (l+1));
    vector <vector <int> > MUTATION( n+1, vector<int> (l+1));
    vector <vector <int> > NEW_GENERATION( n+1, vector<int> (l+1));


    int i, j, k, generation;
    double x1,x2, maxfit, prob;
    vector <double> temp( n+1 );
    vector <double> best_ele( e+1 );
    vector <int> best_position( e+1 );
    int sum=0, p , r, maxpos;
    double avgfit;


    ofstream OUT;
    OUT.open("OUTPUT.txt");

    // generating random population
    cout<<"Initial population : \n";
```

```
    OUT<<"Initial population : \n";
    srand(time(0));
    for(i=1;i<=n;i++){
        cout<<i<<".  ";
        OUT<<i<<".  ";
        for(j=1;j<=l;j++){
            POPULATION[i][j] = (rand()%(2));
            cout<<POPULATION[i][j];
            OUT<<POPULATION[i][j];
        }
        cout<<endl;
        OUT<<endl;
    }
    cout<<endl<<endl;
    OUT<<endl<<endl;

    for(i=1;i<=n;i++){
        for(j=1;j<=l;j++){
            NEW_GENERATION[i][j] = POPULATION[i][j];
        }
    }


    for(generation=1;generation<=max_generation;generation++){

        for(i=1;i<=n;i++){
            for(j=1;j<=l;j++){
                POPULATION[i][j] = NEW_GENERATION[i][j];
            }
        }

        // finding decord value and x value
        for(i=1;i<=n;i++){
            sum=0;
            for(j=1;j<=l1;j++){
                sum=sum+pow(2,5-j)*POPULATION[i][j];
            }
            DECODE_VALUE[i][1] = sum;
            sum=0;
            for(j=l1+1;j<=l;j++){
                sum=sum+pow(2,5+l1-j)*POPULATION[i][j];
            }
            DECODE_VALUE[i][2] = sum;
        }
        for(i=1;i<=n;i++){

```

```
            X[i][1] = xmin+(xmax-xmin)*DECODE_VALUE[i][1]/(pow(2,l1)-1);
            X[i][2] = xmin+(xmax-xmin)*DECODE_VALUE[i][2]/(pow(2,l1)-1);
        }

        // calculating function value to find fitness value
        for(i=1;i<=n;i++){

            x1 = X[i][1];
            x2 = X[i][2];
            FUNCTION_VALUE[i][1] = f(x1,x2);
        }
        for(i=1;i<=n;i++){

            FITNESS[i][1] = -1*FUNCTION_VALUE[i][1]; // to conver minimization problem into maximization
        }

        // Selection of best e solution by Elitism and other by tournament selection for mating pool
        for(i=1;i<=n;i++){
            temp[i] = FITNESS[i][1];
        }
        sort(temp.begin()+1,temp.end());

        // for elitism of e no. of element
        for(i=1;i<=e;i++){
            best_ele[i] = temp[n-i+1];
        }
        for(j=1;j<=e;j++){
            for(i=1;i<=n;i++){
                if(FITNESS[i][1]==best_ele[j]){
                    best_position[j] = i;
                }
            }
        }

        // tranfering elitism solution to new generation
        for(i=1;i<=e;i++){
            p=best_position[i];
            for(j=1;j<=l;j++){

                NEW_GENERATION[e][j] = POPULATION[p][j];
            }
        }
```

```
        //tournament selection of n-e no. of element
        for(i=e+1;i<=n;i++){
            r = 1+(rand()%(n)); // random number between 1 to n
            maxfit=FITNESS[r][1];
            maxpos=r;
            for(j=2;j<=toursize;j++){
                r = 1+(rand()%(n));
                if(FITNESS[r][1]>maxfit){
                    maxfit=FITNESS[r][1];
                    maxpos=r;
                }
            }

            for(j=1;j<=l;j++){
                POOL[i][j] = POPULATION[i][maxpos];
            }
        }

        // CROSS-OVER
        for(i=e+1;i<=n;i=i+2){
            prob=fabs(sin(rand())); // random number between 0 and 1
            if(prob<=pc){
                r = 1+(rand()%(l-1));
                for(j=1;j<=r;j++){
                    CROSSOVER[i][j] = POOL[i][j];
                    CROSSOVER[i+1][j] = POOL[i+1][j];
                }
                for(j=r+1;j<=l;j++){
                    CROSSOVER[i][j] = POOL[i+1][j];
                    CROSSOVER[i+1][j] = POOL[i][j];
                }
            }
            else{
                for(j=1;j<=l;j++){
                    CROSSOVER[i][j] = POOL[i][j];
                    CROSSOVER[i+1][j] = POOL[i+1][j];
                }
            }
        }

vector <vector <double> > X( n+1, vector<double> (no_of_variable+1));
        for(i=e+1;i<=n;i=i+2){
            for(j=1;j<=l;j++){
                MUTATION[i][j] = CROSSOVER[i][j];
            }
```

```
        }
        for(i=e+1;i<=n;i=i+2){
            prob=fabs(sin(rand())); // random number between 0 and 1
            if(prob<=pm){
                r = 1+(rand()%(l-1));
                if(CROSSOVER[i][r]==0){
                    MUTATION[i][r]=1;
                }
                else{
                    MUTATION[i][r]=0;
                }
            }
        }

        for(i=e+1;i<=n;i++){
            for(j=1;j<=l;j++){
                NEW_GENERATION[i][j] = MUTATION[i][j];
            }
        }
    }

    cout<<"Final Population after "<<max_generation<<" generation : \n";
    OUT<<"Final Population after "<<max_generation<<" generation : \n";
    for(i=1;i<=n;i++){
        cout<<i<<".  ";
        OUT<<i<<".  ";
        for(j=1;j<=l;j++){
            cout<<POPULATION[i][j];
            OUT<<POPULATION[i][j];
        }
        cout<<endl;
        OUT<<endl;
    }
    cout<<endl<<endl;
    OUT<<endl<<endl;

    cout<<"Position and Function value of all population : \n";
    cout<<"Population#\tx1\tx2\tf(x1,x2)\n";
    cout<<setprecision(3);
    OUT<<"Position and Function value of all population : \n";
    OUT<<"Population#\tx1\tx2\tf(x1,x2)\n";
    OUT<<setprecision(3);
    for(i=1;i<=n;i++){
        x1 = X[i][1];
```

```
        x2 = X[i][2];
        FUNCTION_VALUE[i][1] = f(x1,x2);
cout<<i<<"\t\t"<<x1<<"\t"<<x2<<"\t"<<FUNCTION_VALUE[i][1]<<endl;
OUT<<i<<"\t\t"<<x1<<"\t"<<x2<<"\t"<<FUNCTION_VALUE[i][1]<<endl;
    }
    cout<<endl<<endl;
    OUT<<endl<<endl;

    p = best_position[1];
    cout<<"The minimum of given problem lies at x1 = "<<X[p][1]<<", x2 = "<<X[p][1]<< "<<FUNCTION_VALUE[p][1]<<".";
    OUT<<"The minimum of given problem lies at x1 = "<<X[p][1]<<", x2 = "<<X[p][1]<< "<<FUNCTION_VALUE[p][1]<<".";

    return 0;
}

double f(double x1, double x2){
    return x1+x2-2*x1*x1-x2*x2+x1*x2;
}
```