Name:	**Navin Kumar Adhikari**
Email:	**navinadhikari@nevada.unr.edu**
Title	**Assignment 1 (CS776)**

Black box optimization part 1

Algorithm Used:

```
Variables: vec1[100], vec2[100], fitness, bestftness
Initialization: vec2[100]    = [00...0],  //used to store best vector that gives best fitness
                vec1[100]  = [00...0], //used for calculating new candidate solution bestfitness

For iteration = 0 to maxitr: step +1

    vec1 <--- nextcanditatesol() //Generate next candidate solution
    fitness <-- eval(vec1)
    if(fitness >bestfitness)
    {
        bestfitness <--- fitness
        vec2 <--- vec1
    }
     else
     {
         vec1 <--- vec2
      }
   End For Loop
```

Algorithm to generate next candidate solution from current best
----

Flip 2-bit randomly from the previous best solution

```
For i = 0 to maxitr step +1
   Index value1 of bit change <--- random number in [0,99]
   Index value2 of bit change <--- random number in [0,99]
   Index value3 of bit change <--- random number in [0,99]
   vec[index value1] <--- vec[index value2] XOR
   vec[index value2] <--- vec[index value 3] XOR
  End For
```

Working of Algorithm

Firstly, the algorithm initializes the solution vectors **vec1[100]** and **vec2[100]** to all zero value. **vec2[100]** is used to store the best solution so far and **vec1[100]** is used for the calculation. The **bestfitness** stores the best fitness value obtained so far and the **fitness** stores the fitness value of the current solution. Then, the first initializes value is passed to the eval function and returned value is stored in fitness and bestfitness. After entering into loop from zero to maxitr, the initialized value in the vec1[100] is passed into the function to generate a next candidate solution. If the fitness of the new a candidate solution is greater or equal to the best fitness then **bestfitness** value is updated with the new fitness value. In addition, the **vec2[100]** stores the **vec1[100]** value as the best solution obtained so far. The algorithm proceeds to the next iteration with this new **vec1[100]** and again check for the fitness with bestfitness. But, if the above two conditions fail then, previous best solution stored in **vec2100]** is copied to **vec1[100]** and the algorithm proceeds with the previous best solution.

<u>For eval.o</u>

The algorithm works perfectly for the first black box function since it might have convex function with minimal space for local maxima and wider space to reach the maximum value. Furthermore, believe that my algorithm works fine for solution space with number of local optimum as long as it has one global optimum that does not have near vertical hills (steepest hills). My algorithm does not work if the optimum solution is in the space has very steepest.

For eval1.o

My above algorithm is unable to get the maximum value of 100. The algorithm is stuck at local maxima of 70. The reason behind this is that the function might have a number of local maxima with only one narrow global maxima with sharply steepest hills.

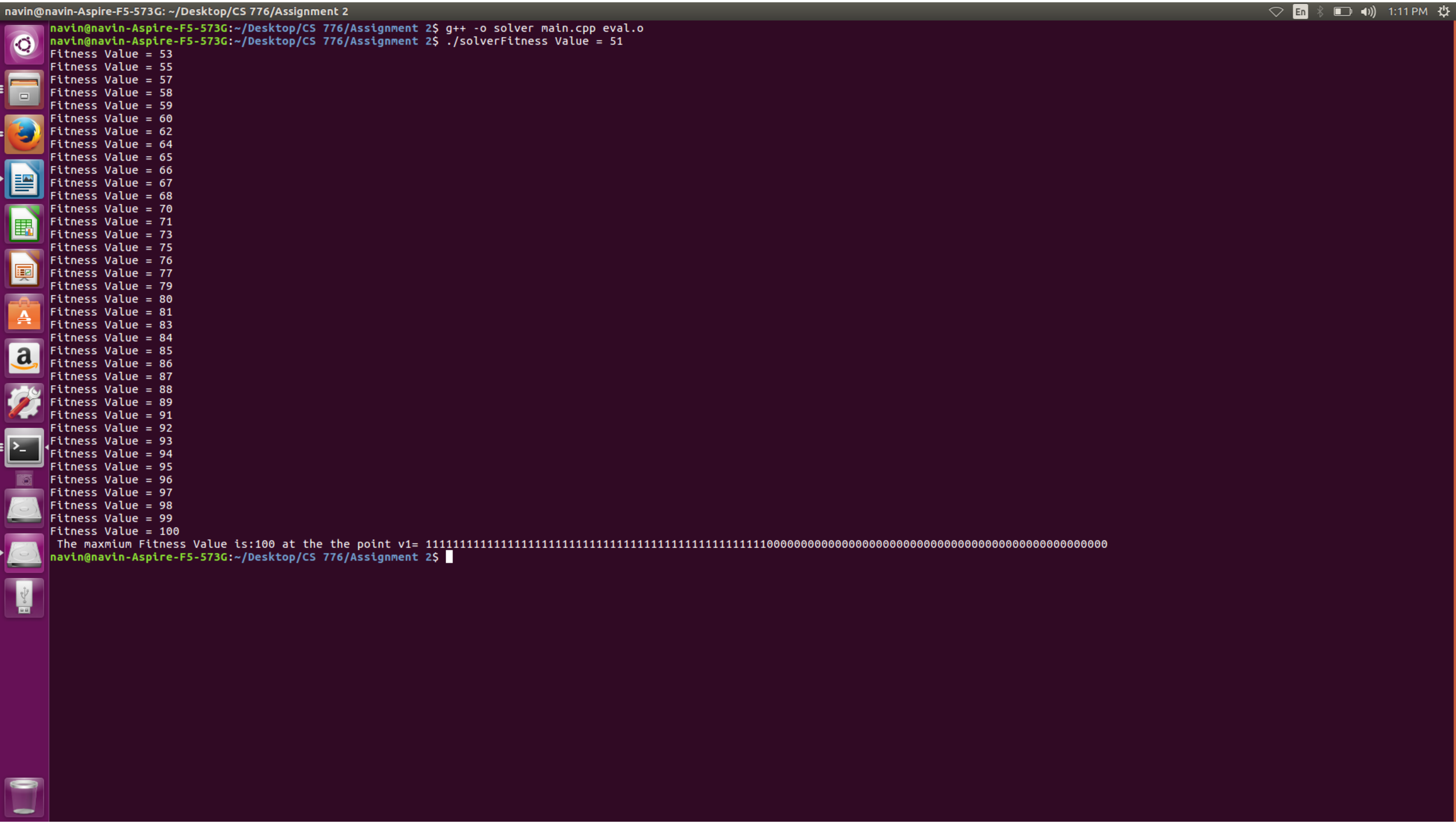
<u>Advantages of Algorithm:</u>
1. Multiple random points to find the global maxima which avoids the condition of being stuck at a local maxima.
2. The cost (Constant factor *base cost) of the climbing of a hill low. Where constant factor is how many times you are willing to randomly restart.

<u>Weakness of this algorithm:</u>

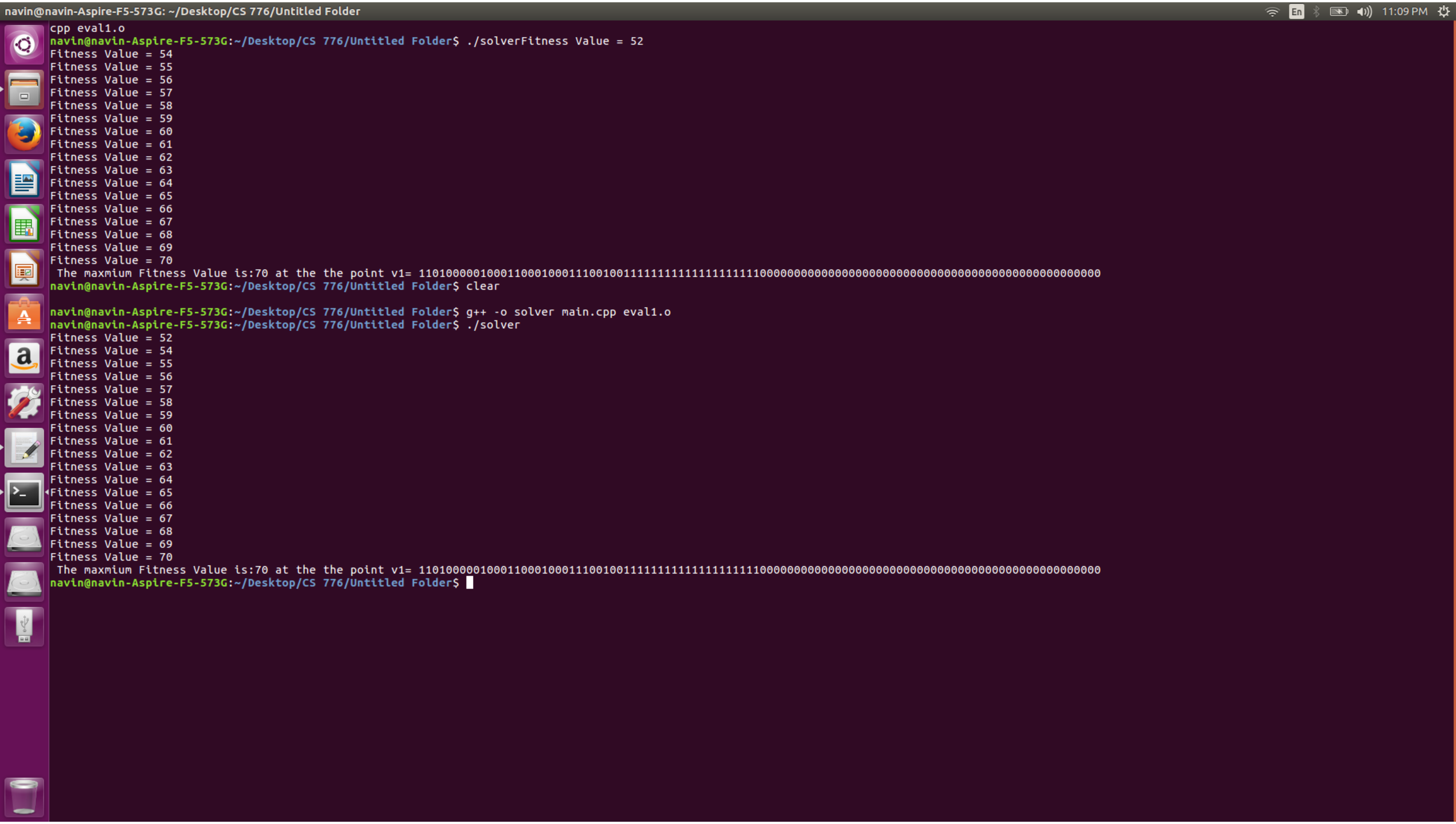
This algorithm cannot search for the solution that is located in a narrow solution space.The reason is that the space where the solution lies is very slim such that the randomly generated candidate almost always falls on the solution space filled with local optimums.

<u>Code for Hill Climber</u>

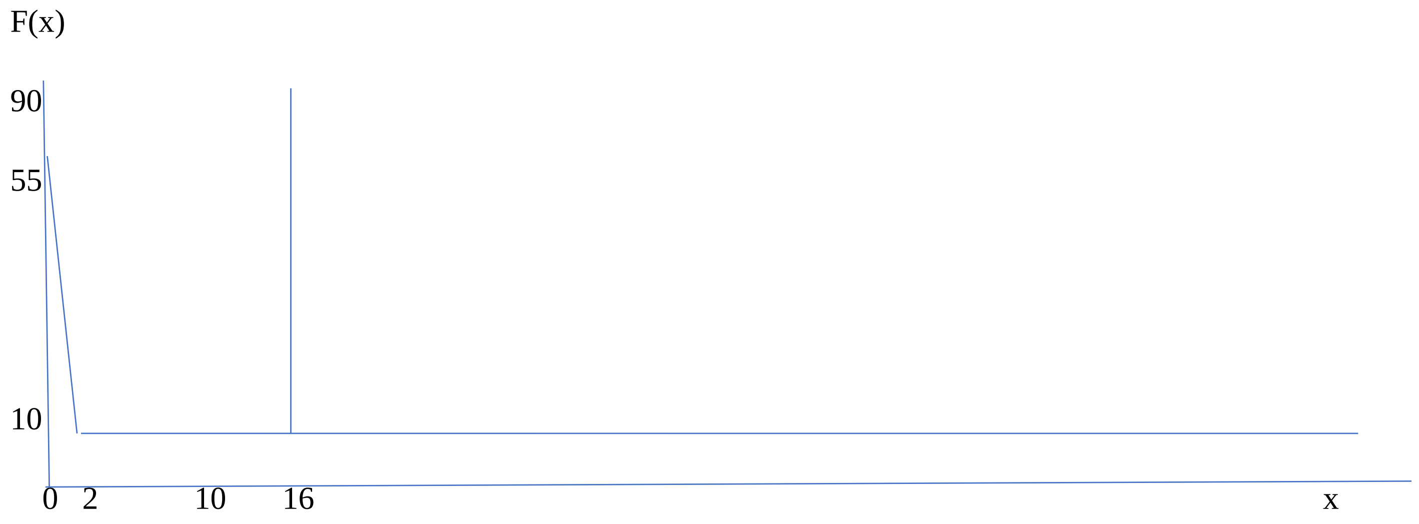
```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

using namespace std;
//declaration of eval function
double eval(int *pj);
// A function that copies the values of array1 into array2.
void copyvector(int arr1[], int arr2[])
{
    for(int i = 0; i < 100; i++)
    {
        arr2[i] = arr1[i];
    }
}
//A function initilizes the start point for hill climbing.
void candidatesol( int arr[],int size)
{

    int randnum;
    for(int i = 0; i < size; i++)
    {
      arr[i] = 0;
    }
}
// A function that randomizes bit of candidate soution to find the next-
//candidate solution.
void nextcanditatesol( int arr[], int size)
{
   for( int i=0;i<1;i++)
   {
       int k1=rand()%100;
       int k2=rand()%100;
       int k3=rand()%100;
       arr[k1]=arr[k2]^1;
       arr[k2]=arr[k3]^1;

   }
}
//Main
int main()
{
  double maxitr=10000;
  const int bitsize=100;
  int vec1[bitsize];
  int vec2[bitsize];
  double fitness=0.0, bestfitness=0.0;
  candidatesol(vec1,bitsize);
  candidatesol(vec2,bitsize);
  fitness = eval(vec1);
  bestfitness=fitness;
  for (int i=0;i<maxitr;i++)
  {
   nextcanditatesol(vec1,bitsize);
   fitness = eval(vec1);
   if(fitness>bestfitness)
   {
```



```
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Assignment 2$ g++ -o solver main.cpp eval.o
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Assignment 2$ ./solverFitness Value = 51
Fitness Value = 53
Fitness Value = 55
Fitness Value = 57
Fitness Value = 58
Fitness Value = 59
Fitness Value = 60
Fitness Value = 62
Fitness Value = 64
Fitness Value = 65
Fitness Value = 66
Fitness Value = 67
Fitness Value = 68
Fitness Value = 70
Fitness Value = 71
Fitness Value = 73
Fitness Value = 75
Fitness Value = 76
Fitness Value = 77
Fitness Value = 79
Fitness Value = 80
Fitness Value = 81
Fitness Value = 83
Fitness Value = 84
Fitness Value = 85
Fitness Value = 86
Fitness Value = 87
Fitness Value = 88
Fitness Value = 89
Fitness Value = 91
Fitness Value = 92
Fitness Value = 93
Fitness Value = 94
Fitness Value = 95
Fitness Value = 96
Fitness Value = 97
Fitness Value = 98
Fitness Value = 99
Fitness Value = 100
 The maxmium Fitness Value is:100 at the the point v1= 11111111111111111111111111111111111111111111111111 00000000000000000000000000000000000000000000000000
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Assignment 2$ 
```



```
navin@navin-Aspire-F5-573G: ~/Desktop/CS 776/Untitled Folder
cpp eval1.o
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Untitled Folder$ ./solverFitness Value = 52
Fitness Value = 54
Fitness Value = 55
Fitness Value = 56
Fitness Value = 57
Fitness Value = 58
Fitness Value = 59
Fitness Value = 60
Fitness Value = 61
Fitness Value = 62
Fitness Value = 63
Fitness Value = 64
Fitness Value = 65
Fitness Value = 66
Fitness Value = 67
Fitness Value = 68
Fitness Value = 69
Fitness Value = 70
 The maxmium Fitness Value is:70 at the the point v1= 1101000000100011000100011100100111111111111111111100000000000000000000000000000000000000000000000000
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Untitled Folder$ clear

navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Untitled Folder$ g++ -o solver main.cpp eval1.o
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Untitled Folder$ ./solver
Fitness Value = 52
Fitness Value = 54
Fitness Value = 55
Fitness Value = 56
Fitness Value = 57
Fitness Value = 58
Fitness Value = 59
Fitness Value = 60
Fitness Value = 61
Fitness Value = 62
Fitness Value = 63
Fitness Value = 64
Fitness Value = 65
Fitness Value = 66
Fitness Value = 67
Fitness Value = 68
Fitness Value = 69
Fitness Value = 70
 The maxmium Fitness Value is:70 at the the point v1= 1101000000100011000100011100100111111111111111111100000000000000000000000000000000000000000000000000
navin@navin-Aspire-F5-573G:~/Desktop/CS 776/Untitled Folder$
```



My eval function in which it is very hard for AI to end up with a optimal solution is shown is figure above. In this kind of function, it is difficult for finding the optimal solution by randomly choosing the starting point because the peak value is only one discrete value and it has only one local maxima with very sharp steepest. As such, it is highly likely for each random search to fall into other space rather than on only one pin point position. This function has minimum value of 10 and maximum value of 90.

```
// 100 bit binary representation of x values are compared with vec values to return its corresponding
//functional value (in between 10 and 90)
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

using namespace std;
//declaration of eval function
double eval(int *vec)
{
   int i,flag=0;
   double returnValue;
         for (i = 5; i < 100; i++)
         {
                  if (vec[i] == 0)
                  {
                           flag = 1;
                  }
                  else
                  {
                           flag = 0;
                           goto stage;
                  }
```

```
        }
           if(flag==1)
           {
           if (vec[0] == 0 && vec[1] == 0 && vec[2] == 0 &&vec[3] == 0 && vec[4] == 0)
                {
                     returnValue = 55;
                }
               else if (vec[0] == 1 && vec[1] == 0 && vec[2] == 0 &&vec[3] == 0 &&vec[4] == 0 )
             {
                  returnValue = 50;
                }
               else if (vec[0] == 0 && vec[1] == 0 && vec[2] == 1&&vec[3] == 1 && vec[4] == 0)
               {
                  returnValue = 90;
               }
            else
             {
               goto stage;
             }
          }

          if (flag==0)
               {
                stage: returnValue = 10;
           }

    }

// A function that copies the values of array1 into array2.
void copyvector(int arr1[], int arr2[])
{
    for(int i = 0; i < 100; i++)
    {
        arr2[i] = arr1[i];
    }
}
//A function initilizes the start point for hill climbing.
void candidatesol( int arr[],int size)
{

    int randnum;
    for(int i = 0; i < size; i++)
    {
      arr[i] = 0;
    }
}
// A function that randomizes bit of candidate soution to find the next-
//candidate solution.
void nextcanditatesol( int arr[], int size)
{
   for( int i=0;i<1;i++)
   {
       int k1=rand()%100;
       int k2=rand()%100;
       int k3=rand()%100;
       arr[k1]=arr[k2]^1;
       arr[k2]=arr[k3]^1;

   }
}
//Main
int main()
{
  double maxitr=10000;
  const int bitsize=100;
  int vec1[bitsize];
  int vec2[bitsize];
  double fitness=0.0, bestfitness=0.0;
  candidatesol(vec1,bitsize);
  candidatesol(vec2,bitsize);
  fitness = eval(vec1);
  bestfitness=fitness;
  for (int i=0;i<maxitr;i++)
  {
   nextcanditatesol(vec1,bitsize);
   fitness = eval(vec1);
   if(fitness>bestfitness)
   {
    bestfitness=fitness;
    copyvector(vec1,vec2);//copies the max value point so far.
    cout << "Fitness Value = " << bestfitness << endl;
   }
  else
   {

    copyvector(vec2,vec1);//Retrun to previous max value point to find new random start point-
                        //if the value halts to certain point making local maxima.
   }
```

```
 }
 cout<<" The maxmium Fitness Value is:"<<bestfitness<<" at the the point v1= ";
  for (int k=0;k<bitsize;k++)
 {
    cout<<vec1[k];
 }
 cout<<endl;
}
```