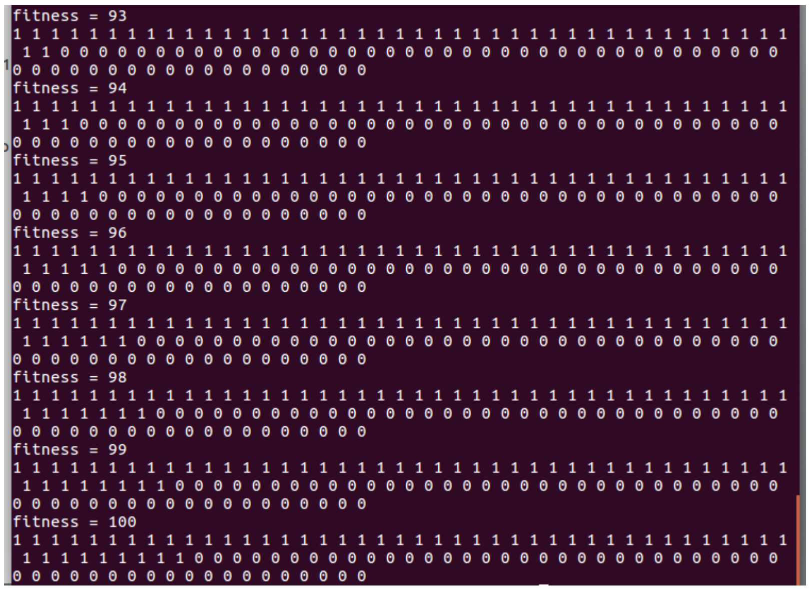
CS 776: Evolutionary Computing -Assignment 1 **Fall 2018**

Full Name: Batyr Charyyev
Email address: bcharyyev@nevada.unr.edu

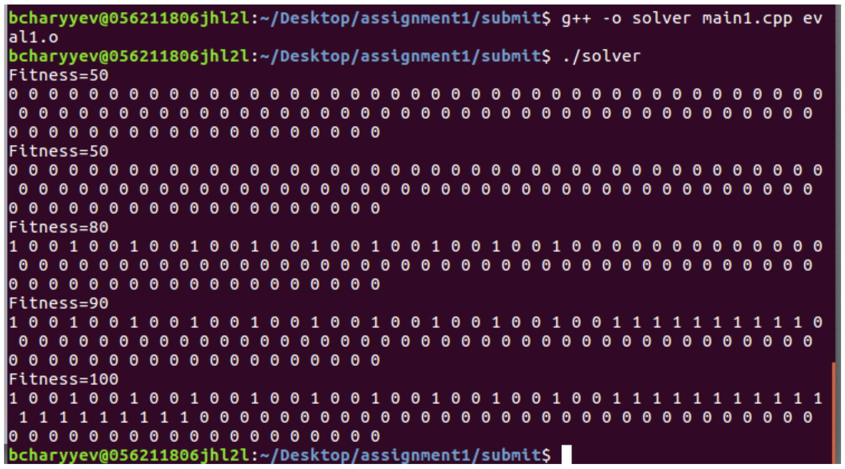
Part-1.

For **eval.o** I used simple hillclimbing by just checking one digit per iteration and comparing result obtained with my old result. In lecture it was mentioned that it is just one hill, so by applying simple hill climbing I was able to get **100**. Strength of this algorithm is it is good for simple functions with one hill and it can find global optimum really quick. Weakness is, it is not good for complex functions with many local optimum because it may stuck on local optimum values.



```
fitness = 93
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 94
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 95
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 96
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 97
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 98
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 99
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
fitness = 100
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

For **eval1.o** first I tried to find pattern because in lecture it was mentioned to look for patterns. I observed that digits between **0-30** and **30-50** and **50-100** have similar behavior. Thus, I tried to split 100 digits into smaller partitions and consider each one of them as one dimension in my search space. I have two variables named **window** and **offset.** Window represents size of my partition and gap between two consecutive 1 digits in that partition, for example if window is 2 in that iteration first 20 digits will be considered as one partition and one 1 digit will be followed by 1 zero digit. Offset represents offset from which that partition starts.



```
bcharyyev@056211806jhl2l:~/Desktop/assignment1/submit$ g++ -o solver main1.cpp ev
al1.o
bcharyyev@056211806jhl2l:~/Desktop/assignment1/submit$ ./solver
Fitness=50
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Fitness=50
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Fitness=80
1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Fitness=90
1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Fitness=100
1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
bcharyyev@056211806jhl2l:~/Desktop/assignment1/submit$ 
```

Strength is it may find optimum values for functions following similar patterns. It may give good results for functions in which some block of digits show similar behaviors, however, it may give really bad results for other functions which doesnt follow this pattern.

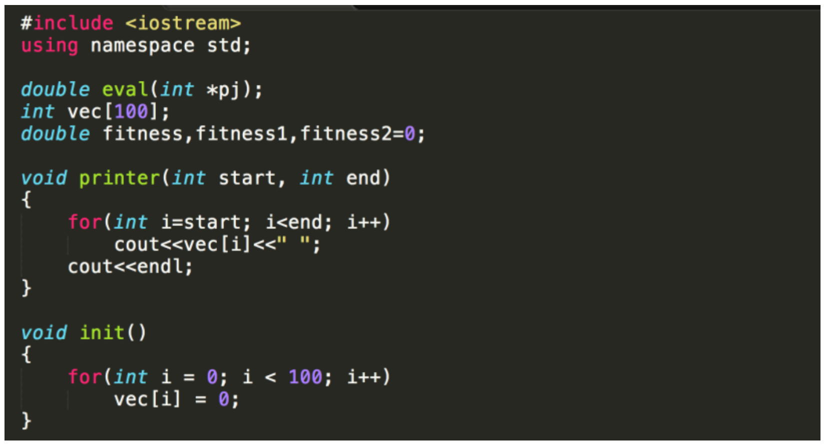
Part-2.

You can use eval.cpp to generate another eval object which is harder to solve. Min and max values are 0 and 100 respectively.
It is hard because I generated random combination which does not follow any pattern thus, does not give any clue. Furthermore, I check 2 consecutive digits at once in each iteration. Thus, it is hard for my hillclimbers to solve it.

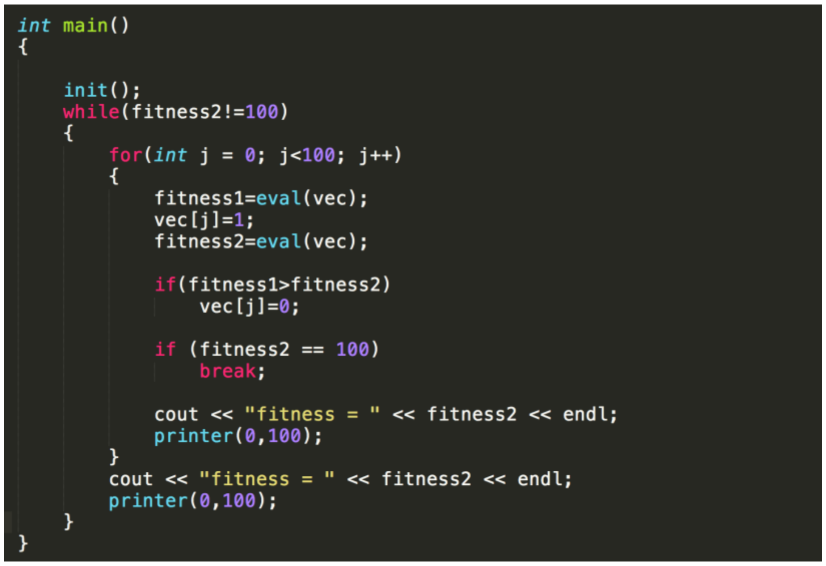
Command: *gcc -Wall -c eval.cpp.*

Source for eval.o

I have loop in which I change one of the digits and check new fitness value with old fitness value. If new fitness if less than older one, I revert change. And I continue this till my fitness is 100.



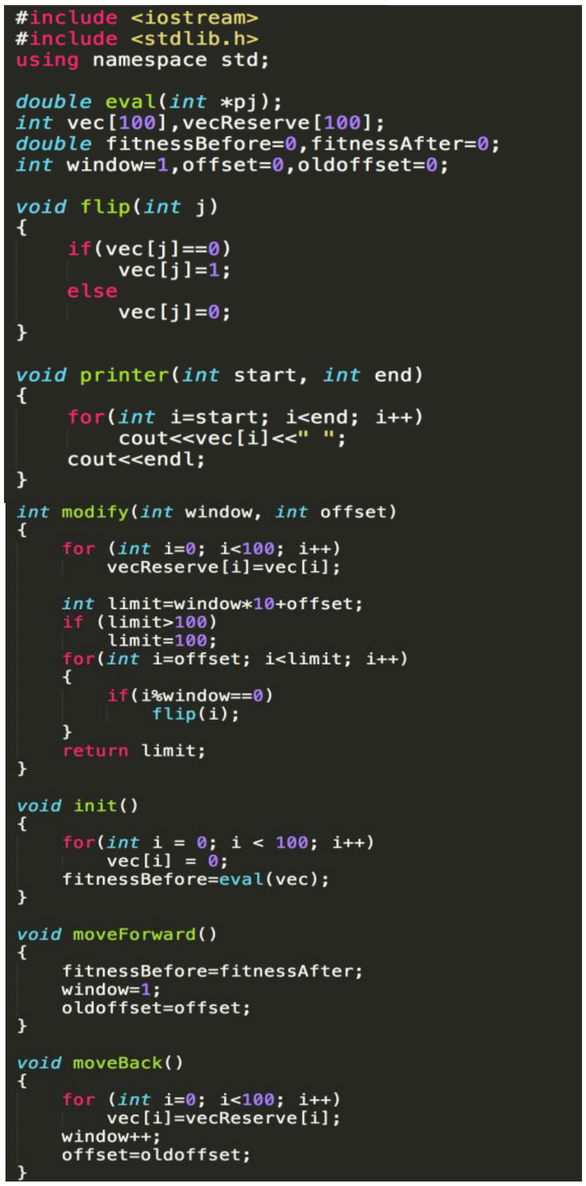
```
#include <iostream>
using namespace std;

double eval(int *pj);
int vec[100];
double fitness,fitness1,fitness2=0;

void printer(int start, int end)
{
    for(int i=start; i<end; i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}

void init()
{
    for(int i = 0; i < 100; i++)
        vec[i] = 0;
}
```



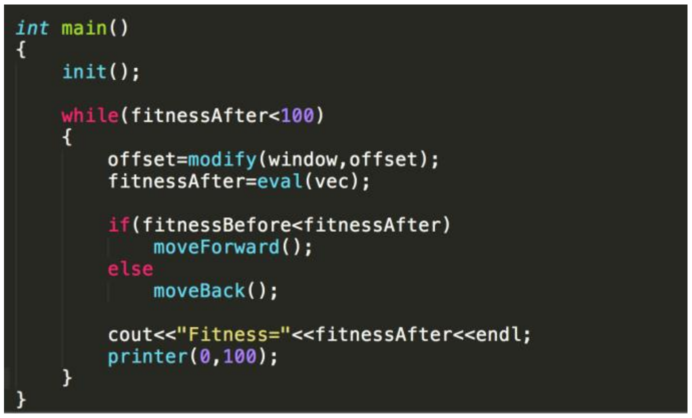
```
int main()
{

    init();
    while(fitness2!=100)
    {
        for(int j = 0; j<100; j++)
        {
            fitness1=eval(vec);
            vec[j]=1;
            fitness2=eval(vec);

            if(fitness1>fitness2)
                vec[j]=0;

            if (fitness2 == 100)
                break;

            cout << "fitness = " << fitness2 << endl;
            printer(0,100);
        }
        cout << "fitness = " << fitness2 << endl;
        printer(0,100);
    }
}
```

Source for eval1.o

Since there is a pattern in eval1, I separate “vec” into partitions and consider each one of them as one dimension in search space. Size depends on variable window and it may change in each iteration based on results obtained from previous iteration. And starting position of partition depends on variable offset which may also change in each iteration based on previous iterations.

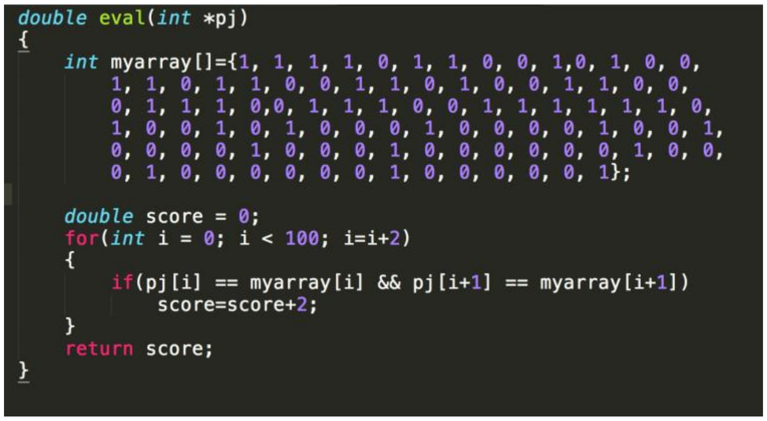


```
#include <iostream>
#include <stdlib.h>
using namespace std;

double eval(int *pj);
int vec[100],vecReserve[100];
double fitnessBefore=0,fitnessAfter=0;
int window=1,offset=0,oldoffset=0;

void flip(int j)
{
    if(vec[j]==0)
        vec[j]=1;
    else
        vec[j]=0;
}

void printer(int start, int end)
{
    for(int i=start; i<end; i++)
        cout<<vec[i]<<" ";
    cout<<endl;
}
int modify(int window, int offset)
{
    for (int i=0; i<100; i++)
        vecReserve[i]=vec[i];

    int limit=window*10+offset;
    if (limit>100)
        limit=100;
    for(int i=offset; i<limit; i++)
    {
        if(i%window==0)
            flip(i);
    }
    return limit;
}

void init()
{
    for(int i = 0; i < 100; i++)
        vec[i] = 0;
    fitnessBefore=eval(vec);
}

void moveForward()
{
    fitnessBefore=fitnessAfter;
    window=1;
    oldoffset=offset;
}

void moveBack()
{
    for (int i=0; i<100; i++)
        vec[i]=vecReserve[i];
    window++;
    offset=oldoffset;
}
```



```
int main()
{
    init();

    while(fitnessAfter<100)
    {
        offset=modify(window,offset);
        fitnessAfter=eval(vec);

        if(fitnessBefore<fitnessAfter)
            moveForward();
        else
            moveBack();

        cout<<"Fitness="<<fitnessAfter<<endl;
        printer(0,100);
    }
}
```

Source for my eval.



```
double eval(int *pj)
{
    int myarray[]={1, 1, 1, 1, 0, 1, 1, 0, 0, 1,0, 1, 0, 0,
        1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0,
        0, 1, 1, 1, 0,0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0,
        1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
        0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1};

    double score = 0;
    for(int i = 0; i < 100; i=i+2)
    {
        if(pj[i] == myarray[i] && pj[i+1] == myarray[i+1])
            score=score+2;
    }
    return score;
}
```