Assignment 1

Tanvir Ahmed

Email: [tanvir\_mclv@gmail.com](mailto:tanvir_mclv@gmail.com)

**Hill Climber Algorithm:**

The algorithm initiates with two vectors. One with the first value 0, rests are 1; other with the first value 1 and rests are 0. The eval() function then compares the two values and set the best value.

When the one value is better than other value, the vector associated with the lower value changes. It changes in a way, where instead of the first value, the first two values changes. At this point, the new vector, is compared with the earlier vector, if the value is same or better, the other vector changes. And, the iteration goes on unless we get the value equal to the max value.

**Strength:**

If the distribution of the data follows a bell curve, then it’s easily possible to get to the top. In the worst case, it have to iterate n/2 times.

**Weakness:**

In the worst case, it might have to iterate through n/2, which is sometimes big. Again, the algorithm does not work, if there is a static value for a long span of data. Also, the algorithm will fail to produce result, when there’ll be any sharp changes in values:

**Hill Climber:**

#include <iostream>

using namespace std;

double eval(int \*pj);

void printme(int vec[]);

double calc(int arr[], int div);

double diff\_calc(int arr1[], int div1);

int main(){

int vec[100];

int best\_vec[100];

double fitness\_v, fitness\_bv;

int divider = 50;

fitness\_v = diff\_calc(vec, divider);

cout << "\*\*\*v" << fitness\_v << endl;

if (fitness\_v == 100){

printme(best\_vec);

return fitness\_v;

}

fitness\_bv = calc(best\_vec, divider);

cout << "\*\*bv\*" << fitness\_bv << endl;

if (fitness\_bv == 100){

printme(best\_vec);

return fitness\_bv;

}

while(fitness\_bv >= fitness\_v and divider < 100){

divider ++;

fitness\_v = calc(vec, divider);

//cout << "\*\*\*\*" << fitness\_v << endl;

}

while(fitness\_v > fitness\_bv and divider < 100){

divider ++;

fitness\_bv = diff\_calc(vec, divider);

//cout << "####" << fitness\_v << endl;

}

}

double calc(int arr[], int div){{

for(int i = 0; i < div; i++){

arr[i] = 1;

}

for(int i = div; i < 100; i++){

arr[i] = 0;

}

}

return eval(arr);

}

double diff\_calc(int arr1[], int div1){{

for(int i = 0; i < div1; i++){

arr1[i] = 0;

}

for(int i = div1; i < 100; i++){

arr1[i] = 1;

}

}

return eval(arr1);

}

void printme(int vec[]){

for(int i=0; i < 100; i++){

cout << vec[i];

}

cout << '\n';

}

Eval function:

#include <iostream>

using namespace std;

string eval(int\* x);

int main (){

string result = eval(x);

}

string eval(int\* x){

int sum = 0;

for (int i = 0; i < 20; i++){

if (i%2 == 0 and x[i] == 1){

sum ++;

}

}

for (int i = 20; i < 50; i++){

if (i%2 == 1 and x[i] == 0){

sum ++;

}

}

if(x[23] and x [45] and x[78]){

sum += 100;

}

if(sum > 100){

return "Well done!";

}

else{

return "Try again";

}

}

**Github:** https://github.com/datracker/Evolutionary-Comp.git