INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Genetic Algorithms

Practica 4:

Profa.: MORALES GUITRON SANDRA LUZ

Grupo: 3CM5

Alumno:

Salcedo Barrón Ruben Osmair.

MEXICO, D.F. a 1 de noviembre del 2018

Introducción

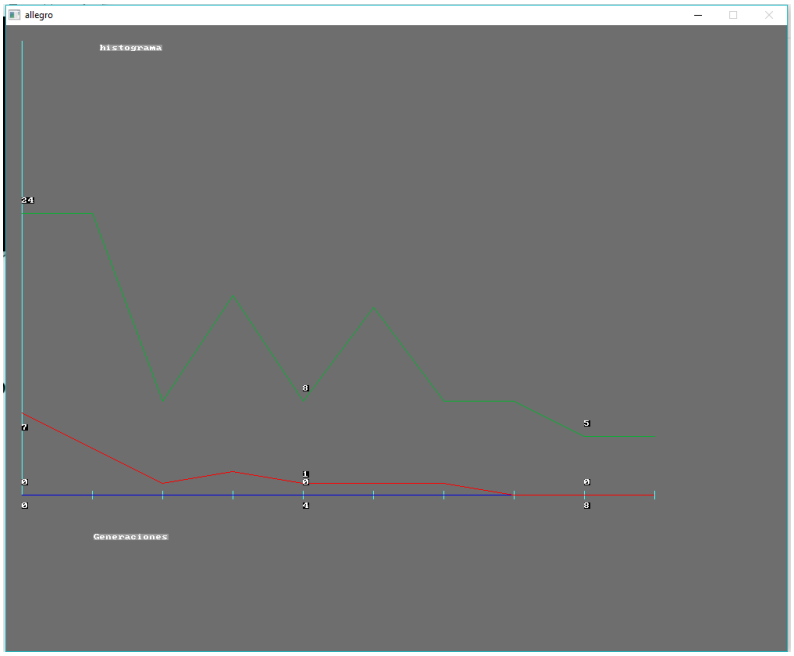
El método de selección por torneo es escoger individuos con base en comparaciones directas entre ellos.

En esta práctica veremos 1 tipo de los 2 posibles de selección mediante torneo: **Determinística, Probabilística**. En esta práctica implementamos la versión probabilística, la cual consiste en barajar los individuos de la población, escoger un numero p de individuos después compararlos con base en sus aptitudes, generar un flip y por último elegir al individuo más apto si flip resulta verdadero, en caso contrario se elige al individuo menos apto.

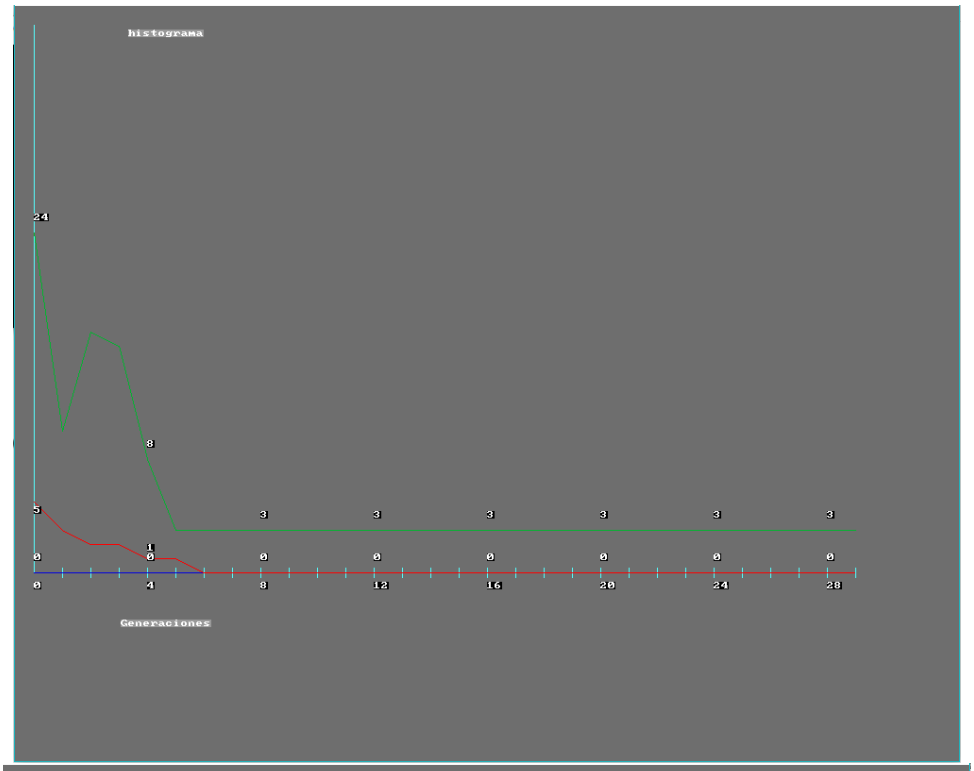
Desarrollo

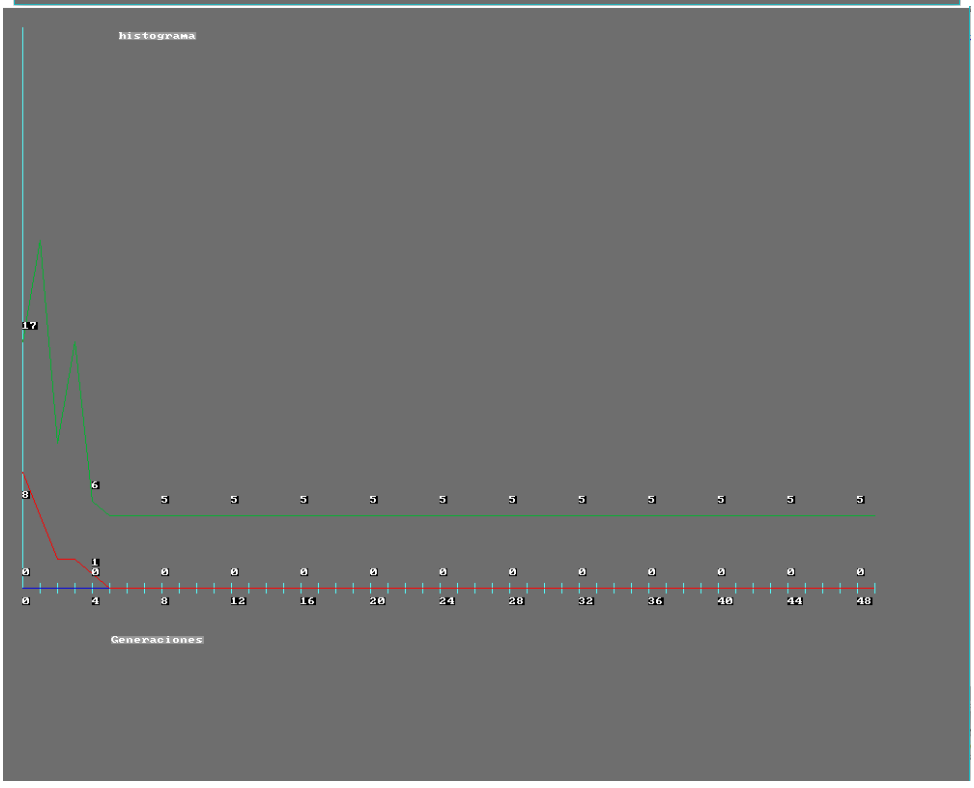
En esta práctica se implementó 4 arreglos de bits para cada etapa en el algoritmo: Población inicial, Población de individuos seleccionados mediante el algoritmo de selección por torneo, Población después de cruza. Población después de mutación.

Al principio se llena aleatoriamente el arreglo de población inicial con series de 5 bits, después para implementar la selección por torneo, se barajean los individuos de la población para así generar parejas para el enfrentamiento. En el primer enfrentamiento, se seleccionan 16 individuos ganadores, se tiene que volver a realizar el algoritmo, para de este modo completar los 32 individuos. Una vez teniendo la población de selección de padres, se realiza una cruza de individuos. Se muestran ejemplos con 10, 30, 50 y 100 generaciones.



GENETIC ALGORITHMS





GENETIC ALGORITHMS

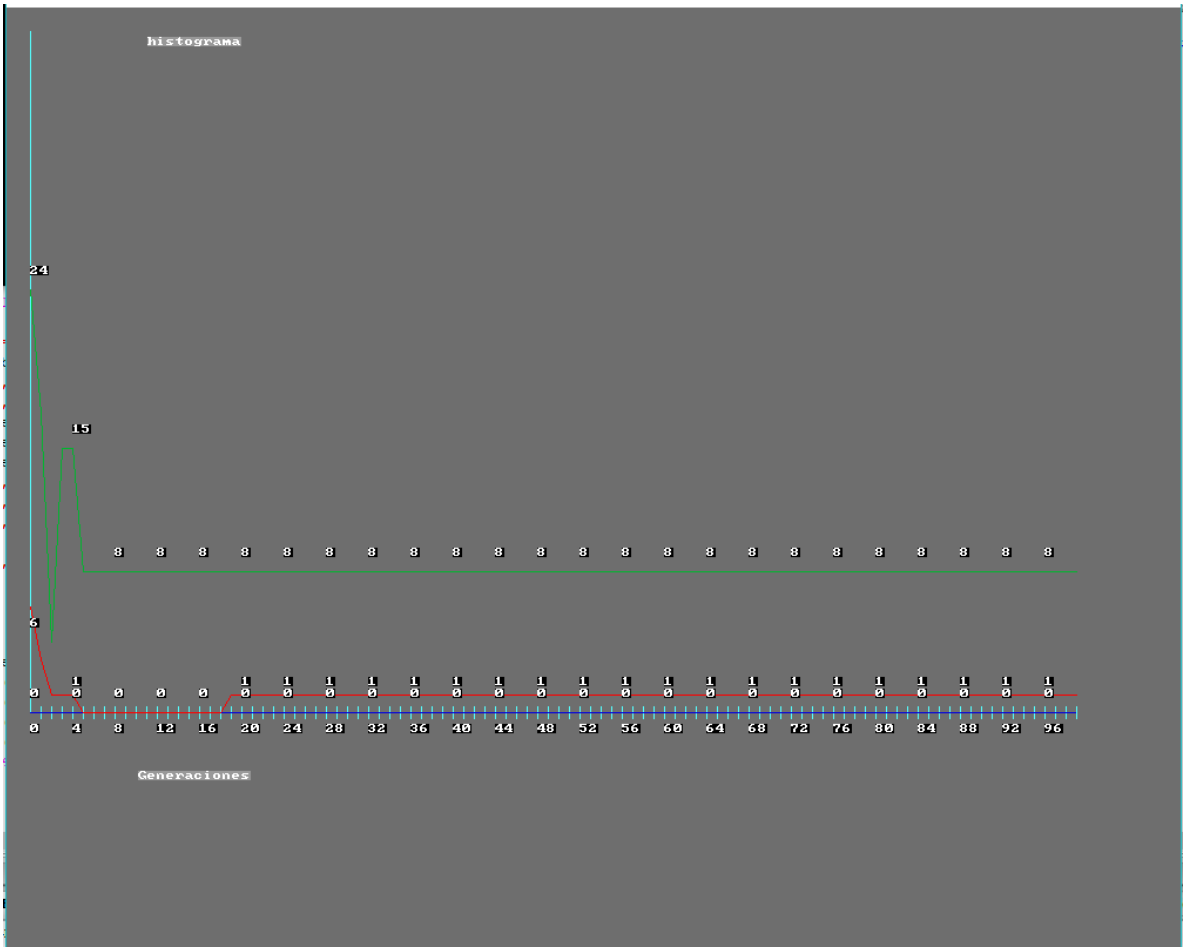


tabla.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

Generacion: 3 totalA:48813600

N	inicial	Aptitud	seleccion	cruza	mutacion
1	00000	2.50	00010	00011	
2	00000	2.50	01011	01010	
3	00000	2.50	01100	01101	
4	00100	0.80	00101	00100	
5	01000	1.00	01000	01000	
6	01000	1.00	00000	00000	
7	00000	2.50	00001	00000	
8	01010	3.43	00000	00001	
9	00001	1.41	01000	01000	
10	01111	3.77	01000	01000	
11	01100	4.78	00000	00000	
12	00000	2.50	00000	00000	
13	00000	2.50	00000	00000	
14	01001	1.66	00000	00000	
15	10010	10.41	10000	10010	
16	01011	6.00	00010	00000	
17	01000	1.00	01010	01000	
18	00000	2.50	01000	01010	
19	00001	1.41	01000	01000	
20	00000	2.50	01000	01000	
21	00010	1.03	00000	00000	
22	00000	2.50	00000	00000	
23	01000	1.00	10000	10000	
24	10000	6.42	00000	00000	
25	00000	2.50	01000	01000	

Conclusiones

La selección probabilística nos dice que los individuos pueden no ser los más aptos, a diferencia de la selección determinista, que siempre nos dará el individuo más apto. Como nos decía en la clase la naturaleza tiene varios factores para determinar un ganador y a veces este ganador no es el más apto y en esta practica pudimos simular algo similar.

Código

main.cpp

```
#include <allegro.h>
#include "inicia.h"
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <bitset>
#include "torneo.h"
#include <iomanip>

using namespace std;
int main ()
{
    int k=0,i=0,h=0,w=0,f=0;
    ofstream tabla;
    tabla.open("tabla.txt", fstream::out);
    int num_generaciones;
    printf("Da el numero de generaciones:");
    scanf("%d",&num_generaciones);

    bitset<BIT_IND> inicial[NUM_IND];
    bitset<BIT_IND> seleccion[NUM_IND];
    bitset<BIT_IND> cruza[NUM_IND];
    bitset<BIT_IND> mutacion_des[NUM_IND];

    int minimo[num_generaciones];
    int maximo[num_generaciones];
    int generationValuesAverage[num_generaciones];
    float ind_apti[NUM_IND];
    float proba[NUM_IND];

    IniciarInd(inicial);
    for ( k = 0; k < num_generaciones; k++){
            w=0;f=0;

            int totalAptitude = getTotalAptitude(inicial);

            for(i=0;i<NUM_BARA;i++){
```

```
                    for(h=0;h<NUM_IND/NUM_BARA;h++){
                         seleccion[w] = tournamentSelection(inicial[f],
inicial[f+1]);
                         w++;
                         f=f+2;
                    }
                    swapInd(inicial);
               }



               for ( i = 0; i < NUM_IND; i+=2){
                    cruza[i] = crossAlgorithm(seleccion[i], seleccion[i +
1]);
                    cruza[i + 1] = crossAlgorithm(seleccion[i + 1],
seleccion[i]);

               }

               for ( i = 0; i < NUM_IND; i++){
                    mutacion_des[i] = cruza[i];
               }



               int mutation_value = NUM_IND / PROBABILITY;

               srand (time(NULL));

               for (i = 0; i < mutation_value; i++){
                    int indivual_to_mutate = rand() % NUM_IND;
                    mutacion_des[indivual_to_mutate] =
mutationAlgorithm(cruza[indivual_to_mutate]);
               }

               for(i=0;i<NUM_IND;i++){
                    ind_apti[i]=(float)getIndividualAptitude(inicial[i]);
               }

               int min_gen_value = getMinGenerationValue(inicial);
               int max_gen_value = getMaxGenerationValue(inicial);
               int gen_average = getGenerationAverage(inicial);

               minimo[k] = min_gen_value;
               maximo[k] = max_gen_value;
               generationValuesAverage[k] = gen_average;

               tabla << "Generacion: " << k+1 <<" "<<"totalA:"<<
totalAptitude<< endl;
               tabla << "N\t\t
inicial\t\tAptitud\t\tseleccion\t\tcruza\t\tmutacion\t\t"<<endl;
               int indice=1;
               for (int i = 0; i < NUM_IND; i++){
                    tabla << indice <<"\t\t"<< inicial[i] <<
"\t\t\t"<<std::fixed<<std::setprecision(2)<<ind_apti[i]<<"\t\t\t" <<
seleccion[i] << "\t\t\t"<< cruza[i] << "\t\t\t" << mutacion_des [i]<<
"\t\t"<< endl;
```

```
                indice++;
             }

            for (int i = 0; i < NUM_IND; i++){
                inicial[i] = mutacion_des[i];
            }

    }

    int sep=900/num_generaciones;
    /*****************************************/

    inicia_allegro(1000,800);
    int in=20;
    BITMAP *buffer = create_bitmap(1000,800);
    clear_to_color(buffer, 0x0a6c92);
    line(buffer, 20, 600, 800, 600,  palette_color[11]);
    line(buffer, 20, 600, 20, 20,  palette_color[11]);
    textout_centre_ex(buffer, font,"histograma", 160, 25, 0xFFFFFF,
0x999999);
    textout_centre_ex(buffer, font,"Generaciones", 160, 650, 0xFFFFFF,
0x999999);
    for(i=0;i<num_generaciones-1;i++){
        line(buffer, in, 600-(minimo[i]*15), in+sep, 600-
(minimo[i+1]*15),  0xbde4ff);
        line(buffer, in, 600-(maximo[i]*15), in+sep, 600-
(maximo[i+1]*15),  0xe5ffdc);
        line(buffer, in, 600-(generationValuesAverage[i]*15), in+sep,
600-(generationValuesAverage[i+1]*15),  0xe5b0dc);

        line(buffer, in+sep, 605, in+sep, 595,  palette_color[11]);
        in=in+sep;
    }
    in=20;
    for(i=0;i<num_generaciones;i+=4){
         textprintf(buffer, font, in, 580-(minimo[i]*15), 0xFFFFFF,
"%d",(minimo[i]));
         textprintf(buffer, font, in, 580-(maximo[i]*15), 0xFFFFFF,
"%d",(maximo[i]));
         textprintf(buffer, font, in, 580-
(generationValuesAverage[i]*10), 0xFFFFFF,
"%d",(generationValuesAverage[i]));
         textprintf(buffer, font, in, 610, 0xFFFFFF, "%d",(i));
        in=in+(sep*4);
    }
    blit(buffer, screen, 0, 0, 0, 0, 1000, 800);
    readkey();

    destroy_bitmap(buffer);

    return 0;
}
END_OF_MAIN ()
```

Inicia.cpp

```
#include "inicia.h"
#include <allegro.h>

void inicia_allegro(int ANCHO_ , int ALTO_){
   allegro_init();
   install_keyboard();

   set_color_depth(32);
   set_gfx_mode(GFX_AUTODETECT_WINDOWED, ANCHO_, ALTO_, 0, 0);
}

int inicia_audio(int izquierda, int derecha){
    if (install_sound(DIGI_AUTODETECT, MIDI_AUTODETECT, NULL) != 0) {
       allegro_message("Error: inicializando sistema de sonido\n%s\n",
allegro_error);
       return 1;
    }

    set_volume(izquierda, derecha);
    return 0;
}
```

torneo.cpp

```
#include "torneo.h"
#include <iostream>
#include <time.h>
#include <stdio.h>
#include <array>
#include <algorithm>
#include <vector>
using namespace std;

void IniciarInd(bitset<BIT_IND> array[]){
    srand (time(NULL));
    for (int i = 0; i < NUM_IND; i++)
            array[i] = 1+rand()%(101-1);
}
void printIndividuals(bitset<BIT_IND> array[]){
    for (int i = 0; i < NUM_IND; i++)
        cout << array[i].to_ulong() << endl;
}

int getIndividualValue(bitset<BIT_IND> individual){
    return individual.to_ulong();
}

float getIndividualAptitude(bitset<BIT_IND> individual){
    float result;
    float numerador;
    float denominador;
    float x = individual.to_ulong();
```

```
    float senX = sin(x);

    numerador = x - 5;
    denominador = 2 + senX;

    result = abs(numerador/denominador);
    //printf("%.2f /%.2f ==%.2f\n",numerador,denominador,result);
return result;
}

float getTotalAptitude(bitset<BIT_IND> array[]){
    float total = 0;
    for (int i = 0; i < NUM_IND; i++)
    {
        total += getIndividualAptitude(array[i]);
    }
    return total;
}
float getProbability(bitset<BIT_IND> individual,int totalAptitude){
    float aux;
    aux=(float)(getIndividualAptitude(individual)/(float)totalAptitude);
   // printf("%.2f / %i
=%.2f\n",getIndividualAptitude(individual),totalAptitude,aux);
     return aux;
}


bitset<BIT_IND> rouletteSelection(bitset<BIT_IND> array[], int
totalAptitude) {

    int r = rand() % (totalAptitude + 1);
    int add = 0;
    int i;
    for(i = 0; i < NUM_IND && add < r; i++) {
        add += getIndividualAptitude(array[i]);
    }
    return array[i];
}
void swapInd(bitset<BIT_IND> set[]){
    array<int, NUM_IND> arr;
    bitset<BIT_IND> aux[NUM_IND];
    for (int i = 0; i < NUM_IND; i++){
        arr[i] = set[i].to_ulong();
    }
    random_shuffle(arr.begin(), arr.end());

    for (int i = 0; i < NUM_IND; i++){
        set[i] = arr[i];
    }
}
bitset<BIT_IND> tournamentSelection(bitset<BIT_IND> &p1, bitset<BIT_IND>
&p2) {

float p = (float)(rand()%NUM_RAND) /(float)NUM_RAND ;
    printf("%.2f\n",p);
    if((getIndividualAptitude(p1) < getIndividualAptitude(p2)) && p >=
0.70){
```

```
        return p2;
    }
    else if ((getIndividualAptitude(p1) < getIndividualAptitude(p2)) && p
< 0.70){
        return p1;
    }
    else if ((getIndividualAptitude(p1) > getIndividualAptitude(p2)) && p
>= 0.70){
        return p1;
    }
    else{
        return p2;
    }
}


bitset<BIT_IND> crossAlgorithm(bitset<BIT_IND> &p1, bitset<BIT_IND> &p2)
{

    bitset<BIT_IND> aux = p1;

    for (int i = 0; i <= PUNTO_CRUZA; i++)
    {
        aux.set(PUNTO_CRUZA - i, p2[PUNTO_CRUZA - i]);
    }

    return aux;
}

bitset<BIT_IND> mutationAlgorithm(bitset<BIT_IND> individual){
    bitset<BIT_IND> result = individual;

    int cont = 0;

    while(cont <= MAX_SEARCH_VALUE)
    {

        int mutation_point = rand() % BIT_IND;
        if(result[mutation_point] == 0){
            result.set(mutation_point, 1);
            break;
        }
        cont++;
    }

    return result;
}

int getMinGenerationValue(bitset<BIT_IND> array[]){
    int min = 1000000, aux = 0;

    for (int i = 0; i < NUM_IND; i++)
    {
        aux = getIndividualAptitude(array[i]);

        if(aux < min){
            min = aux;
```

```
        }

    }
    return min;
}

int getMaxGenerationValue(bitset<BIT_IND> array[]){
    int max = 0, aux = 0;

    for (int i = 0; i < NUM_IND; i++)
    {
        aux = getIndividualAptitude(array[i]);

        if(aux > max){
            max = aux;
        }

    }
    return max;
}

int getGenerationAverage(bitset<BIT_IND> array[]){
    return (getTotalAptitude(array)/NUM_IND);
}
```