Instituto Politécnico Nacional

Escuela Superior de Cómputo

Genetic Algorithms

Reporte Quinta Práctica:
“Selección de individuos por el método de Jerarquías”

Grupo 3CM5
Romero Godinez José Enrique

Introducción:

El algoritmo de selección de individuos por jerarquía consiste en un ajuste de los valores probabilísticos de cada uno de los individuos con el fin de igualar sus oportunidades de ser escogidos en una etapa de selección, esto se hace con un modelo matemático el cuál toma individuos previamente jerarquizados aleatoriamente y les asigna un valor de probabilidad acorde a su jerarquía en la generación. Se toman dos valores de referencia los cuáles serán el mínimo y máximo valor de probabilidad que los individuos pueden tener, Baker recomienda que el valor máximo sea 1.1, debido a que es el que maximiza los resultados en la población.[1]

Objetivo:

Desarrollar un programa en lenguaje c que haga un experimento de selección con 10,30,50 y 100 generaciones de individuos utilizando el método de selección por jerarquías, cruza por punto de cruza y mutación de cambio de bit al 30% de la población.

Solución:

Dado que el lenguaje c no permite de manera nativa las interfaces gráficas se hará uso del programa gnuplot[2] de linux, el cuál será invocado por un programa escrito en lenguaje c que será el encargado de llevar a cabo la selección, cruza y mutación de los individuos para finalmente graficar a los individuos más y menos aptos de cada generación.

Implementación:

Codigo del programa en c:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

void generaIndividuos(unsigned char*);

void seleccionaPadres(unsigned char*,float*);

void calculaDatos(unsigned char*,float*,float*);

int dimeMaximo(unsigned char*);

int dimeMinimo(unsigned char*);

void cruzar(unsigned char*);

void mutar(unsigned char*);

void ordena(unsigned char*);

int main()

{
```

```
unsigned char* individuos=(unsigned char*)malloc(sizeof(unsigned char)*16);
float* aptitud=(float*)malloc(sizeof(float)*16);
float* probabilidad=(float*)malloc(sizeof(float)*16);
int mayor,menor;
float mayores[100];
float menores[100];
generaIndividuos(individuos);
FILE * archivoMaximos = fopen("puntosMaximos.txt", "w");
FILE * archivoMinimos = fopen("puntosMinimos.txt", "w");
for(int i=0;i<100;i++)
{
	calculaDatos(individuos,aptitud,probabilidad);
	mayor=dimeMaximo(individuos);
	mayores[i]=aptitud[mayor];
	menor=dimeMinimo(individuos);
	menores[i]=aptitud[menor];
	//Aqui se guardan los datos a graficar
	fprintf(archivoMaximos, "%d %f \n",i+1,mayores[i]);
	fprintf(archivoMinimos, "%d %f\n",i+1,menores[i]);
	//Guardar datos a graficar
	seleccionaPadres(individuos,probabilidad);
	cruzar(individuos);
	mutar(individuos);

}
char * configGnuplot[] = {"set title \"Histograma\"",
		"set ylabel \"----Aptitud--->\"",
		"set xlabel \"----Generaciones--->\"",
		"set style data histogram",
```

```
                    "set style histogram cluster gap 1",
                        "plot \"puntosMaximos.txt\" using 1:2 with linespoints ,\"puntosMinimos.txt\" using 1:2 with linespoints"
                    };

    FILE * ventanaGnuplot = popen ("gnuplot -persist", "w");
    // Ejecuta los comandos de configGnuPlot 1 por 1
    for (int k=0;k<6;k++){
      fprintf(ventanaGnuplot, "%s \n", configGnuplot[k]);
    }
    fclose(archivoMinimos);
    fclose(archivoMaximos);
    //free(individuos);
    free(aptitud);
    free(probabilidad);
        return 0;
}
void calculaDatos(unsigned char* individuos,float* aptitud,float* probabilidad)
{

        double radianes=3.1416/180;
        for(int i=0;i<16;i++)
        {
                aptitud[i]=(float)abs((individuos[i]-5)/(2 + sin((double)individuos[i]*radianes)));
        }

        for(double j=0;j<16;j++)
        {
                probabilidad[(int)j]=(0.9)+((1.1-0.9)*(j/15));
```

```
	}
}
void seleccionaPadres(unsigned char* individuos, float* probabilidad)
{
	unsigned char* seleccion=(unsigned char*)malloc(sizeof(unsigned char)*16);
	long ltime;
	float valor;
	float suma=0;
	ltime=time(NULL);
	int a=0,stime;
	stime=(unsigned)ltime/2;
	srand(stime);
	for(int i=0;i<16;i++)
	{
		valor=((float)rand()/RAND_MAX)*(1.1-0);

		while(suma<valor)
		{
			suma+=probabilidad[a];
			a++;
		}
		seleccion[i]=individuos[a];
		a=0;
		suma=0;
	}

	free(individuos);
	individuos=seleccion;
```

```
}

void generaIndividuos(unsigned char* individuos)
{
	long ltime;
	ltime=time(NULL);
	int a,stime;
	stime=(unsigned)ltime/2;
	srand(stime);
	for(int i=0;i<16;i++)
	{
		individuos[i]=(unsigned char)(1+rand()%(32-1));
	}
	ordena(individuos);

}
int dimeMaximo(unsigned char* individuos)
{
	int aux=0;
	for(int i=1;i<16;i++)
	{
		if(individuos[aux]<individuos[i])
		{
			aux=i;
		}
	}
	return aux;
}
int dimeMinimo(unsigned char* individuos)
```

```
{
	int aux=0;
	for(int i=1;i<16;i++)
	{
		if(individuos[aux]>individuos[i])
		{
			aux=i;
		}
	}
	return aux;
}
void cruzar(unsigned char* individuos)
{
	long ltime;
	ltime=time(NULL);
	int stime;
	stime=(unsigned)ltime/2;
	srand(stime);
	int puntoCruza=0;
	unsigned char* descendencia=(unsigned char*)malloc(sizeof(unsigned char)*16);
	int hijo=0;
	unsigned char bits,valor;
	puntoCruza=rand()%(5);
	for(int i=0;i<16;i++)
	{
		bits=((unsigned char)(pow((double)2,(double)puntoCruza)));
		if(hijo==0)
		{
			valor=bits&individuos[i];
```

```
                descendencia[i]=(individuos[i+1]-bits)+valor;
                hijo++;
            }
            if(hijo==1)
            {
                valor=bits&individuos[i];
                descendencia[i]=(individuos[i-1]-bits)+valor;
                hijo=0;
                puntoCruza=rand()%(5);
            }
        }
        free(individuos);
        individuos=descendencia;
}
void mutar(unsigned char* individuos)
{
        long ltime;
        unsigned char bits,valor;
        int stime;
        int mutados[4],puntoCruza;

        ltime=time(NULL);
        stime=(unsigned)ltime/2;
        srand(stime);
        mutados[0]=rand()%(15);

        mutados[1]=rand()%(15);

        mutados[2]=rand()%(15);
```

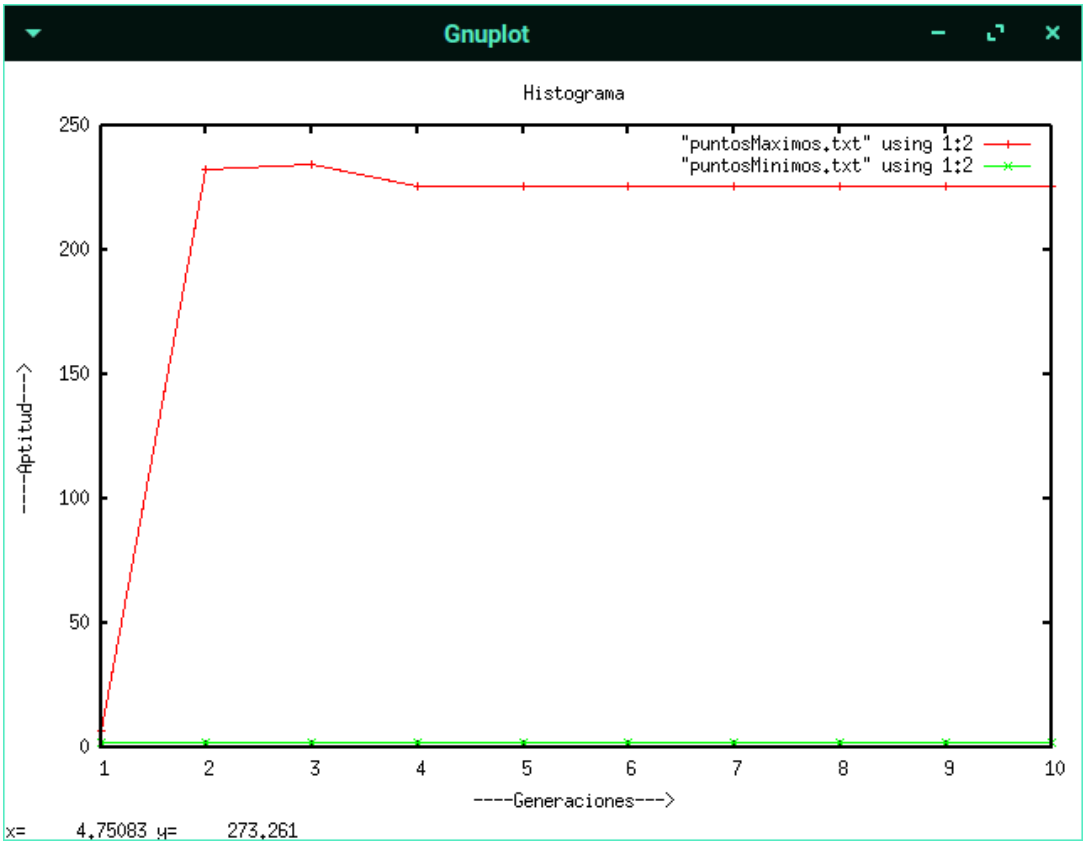
```
	mutados[3]=rand()%(15);


	puntoCruza=rand()%(5);
	bits=((unsigned char)(pow((double)2,(double)puntoCruza)));
	for(int i=0;i<4;i++)
	{
		valor=bits&individuos[mutados[i]];
		if(valor==0)
		{
			individuos[mutados[i]]=individuos[mutados[i]]+bits;
		}else{
			individuos[mutados[i]]=individuos[mutados[i]]-bits;
		}
	}
	ordena(individuos);
}
void ordena(unsigned char* individuos)
{
	unsigned char temp=0;
	for(int i=0;i<16;i++)
	{
		for(int j=0;j<16;j++)
		{
			if(individuos[j]>individuos[j+1])
			{
				temp=individuos[j];
				individuos[j]=individuos[j+1];
				individuos[j+1]=temp;
			}
```

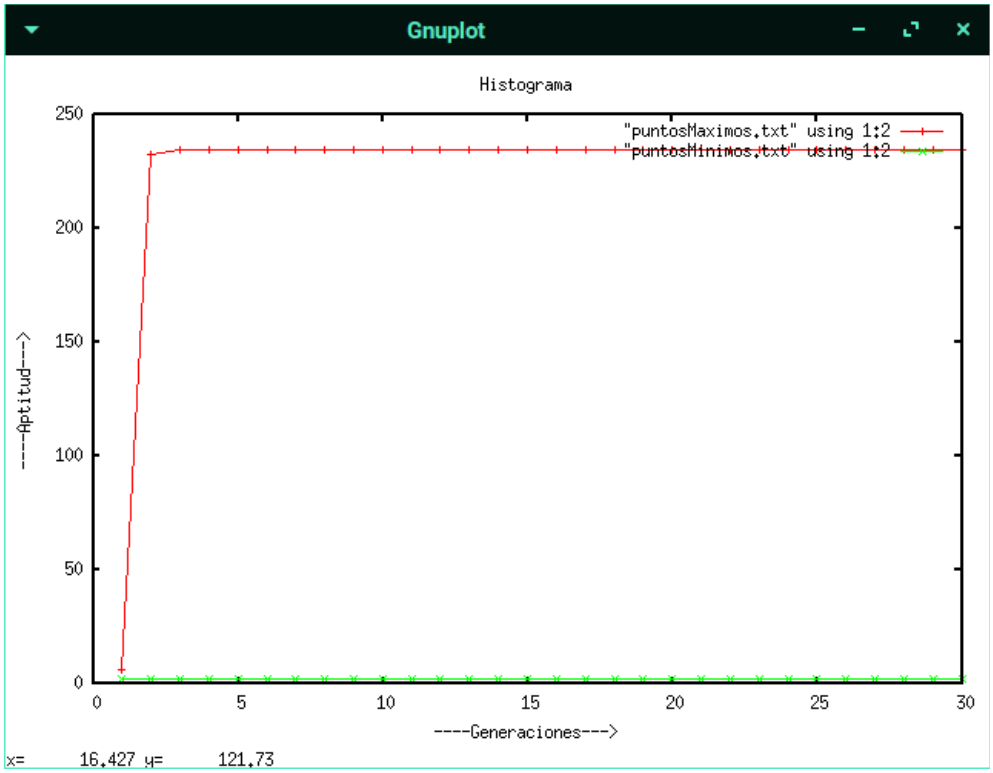
```
            }
        }
}
```

Ejecucion del programa:

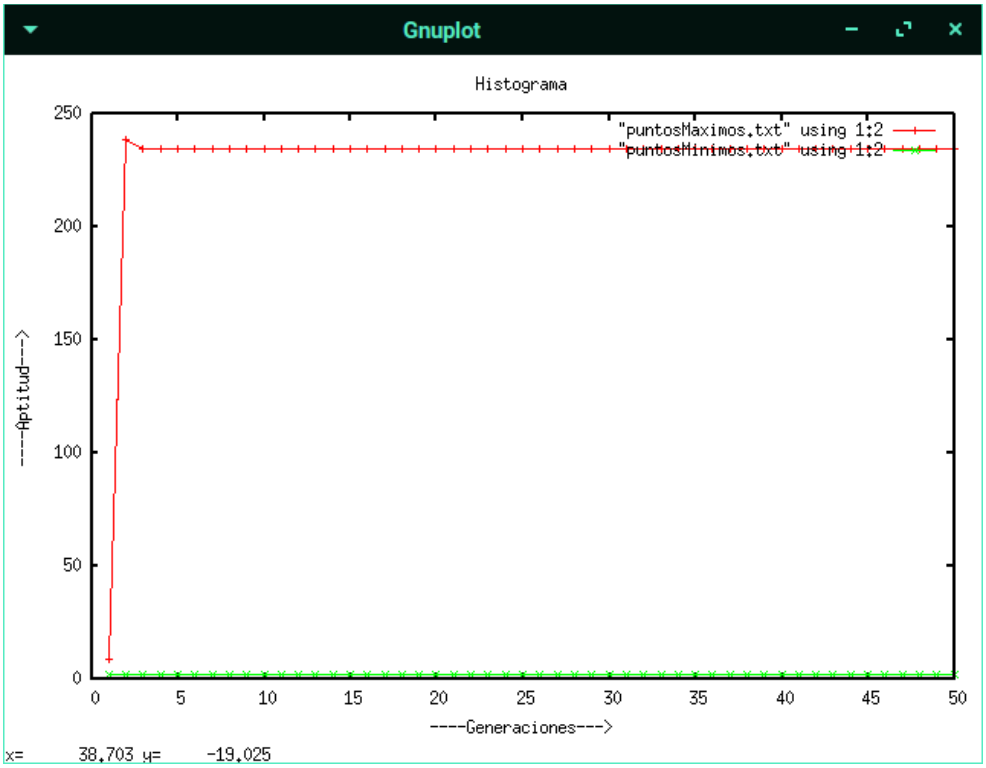
Experimento con 10 generaciones de individuos:



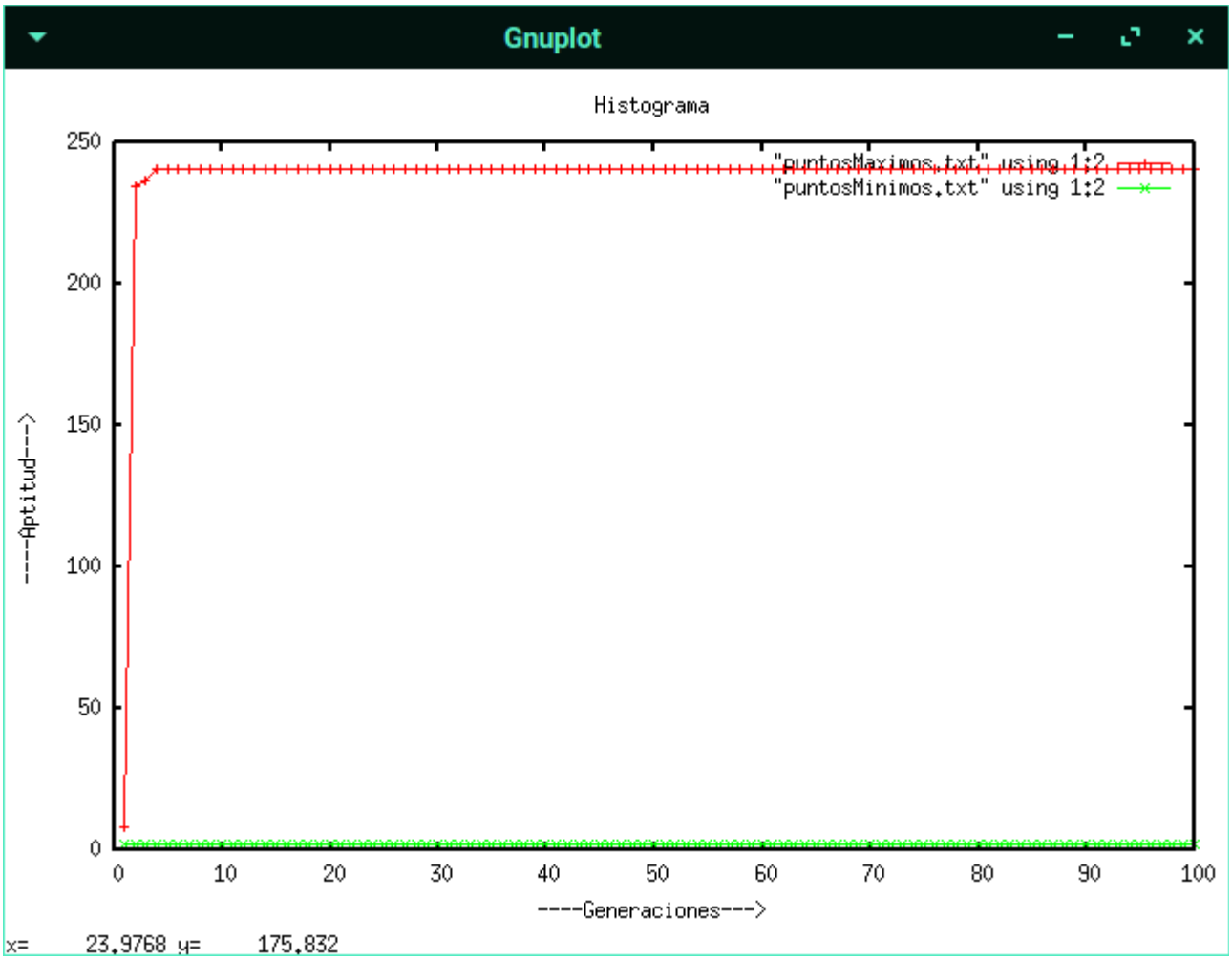
Experimento con 30 generaciones:



Experimento con 50 generaciones:



Experinento con 100 generaciones:



Conclusiones:

Este algoritmo se encarga de normalizar todas las probabilidades de los individuos acorde a lo que el usuario considere una buena jerarquización(a mayor aptitud, mayor ranking, etc.) por lo que le permite a la población converger hacía un valor máximo de manera muy rápida, por lo que a mi consideración es uno de los mejores algoritmos ya que el rango de valores que caen en la “ruleta” tienen casi las mismas posibilidades de caer y como consecuencia la población tiende a quedarse con individuos de una sola clase.

Referencias:

[1]Introducción al Cómputo Evolutivo, Coello