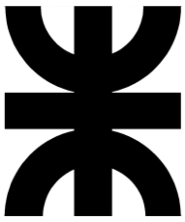
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL ROSARIO
INGENIERÍA EN SISTEMAS DE INFORMACIÓN



TRABAJO PRÁCTICO Nº 1

Cátedra: Algoritmos Genéticos

Integrantes	Legajo
Albizuri, Gastón	40412
Belletti, Kristal	40568
Giordano, Nicolás	40467

Índice

Pág.

Introducción

*Los **Algoritmos Genéticos (AGs)** son algoritmos de búsqueda, optimización y máquinas de aprendizaje basados en la selección natural, genética y el neodarwinismo.*

Fueron desarrollados por John Holland a principios de los años 70. Estos algoritmos constan de una población inicial de individuos, y también de un conjunto de operadores.

- ✓ *Selección: selecciona los individuos que se van a reproducir, en base a su función de aptitud.*
- ✓ *Cruza: intercambia el material genético de dos individuos para generar dos nuevos individuos.*
- ✓ *Mutación: modifica el cromosoma de un individuo que es seleccionado de manera aleatoria.*

A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real.

La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Un algoritmo genético consiste en una función matemática o una rutina de software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia para la nueva generación.

Problemática

Hacer un programa que utilice un Algoritmo Genético Canónico para buscar un máximo de la función:

F(x) = (x/coef)^2 en el dominio [0; 2^{30}-1] donde ***coef = 2^{30}-1***.

Teniendo en cuenta los siguientes datos:

- Probabilidad de Crossover = 0,75
- Probabilidad de Mutación = 0,05
- Población Inicial: 10 individuos
- Ciclos del programa: 20
- Método de Selección: Ruleta
- Método de Crossover: 1 Punto
- Método de Mutación: invertida

El programa debe mostrar, finalmente, el Cromosoma correspondiente al valor máximo obtenido y gráfico, usando EXCEL, de Máx. Mín. y Promedio de la función objetivo por cada generación.

Resolución

El **Algoritmo Genético Simple (AGS)**, también denominado **Canónico**, necesita una codificación o representación del problema, que resulte adecuada al mismo.

Consiste de las siguientes etapas:

1. Generar una población aleatoria de n individuos con una codificación binaria de l-bits.
2. Calcular la aptitud de cada individuo.
3. Seleccionar a los individuos probabilísticamente en base a su aptitud.
4. Aplicar operadores genéticos (cruza y mutación) para generar la siguiente población.

Se repiten las etapas 2, 3, y 4 hasta que cierta condición se cumpla, o se cumpla cierto número m de generaciones (20 veces).

PASOS

1) **Codificación binaria:** La codificación utilizada para el individuo (un cromosoma) de 30 genes sigue el formato de una cadena binaria (0s y 1s), que se generan de forma aleatoria.

2) **Decodificación:** La decodificación de binario a decimal, a partir de una función que toma esa cadena binaria y la transforma en un número decimal.

3) **Evaluación en la función:** Se toma el valor del individuo decodificado y se evalúa en la función a optimizar.

4) **Fitness de cada individuo:** A partir de la sumatoria de todos los valores obtenidos de haber evaluado cada cromosoma en la función a optimizar, se calcula para cada individuo su Fitness: la relación entre la evaluación del individuo en la función y dicha sumatoria.

5) **Ruleta:** Para construirla, a cada cromosoma se le asigna un arco de circunferencia proporcional a su Fitness, los arcos de circunferencias son consecutivos, y el inicio del arco está determinado por la sumatoria de todos los fitness de los cromosomas que lo

anteceden; el final del mismo se encuentra definido por los anterior nombrados y sumando el Fitness correspondiente al cromosoma representante del arco.

6) **Selección por ruleta:** Generamos un número aleatorio, comparamos el numero con el inicio y fin de cada arco de circunferencia, si pertenece al arco de circunferencia se guarda en un arreglo de enteros, denominado elegido, el índice correspondiente al cromosoma representante del arco donde cayó el número aleatorio.

7) **Crossover (Cruza de un punto):** Se toma un par de cromosomas obtenidos en la selección por ruleta, y se consulta la probabilidad de crossover (determinada por un número aleatorio generado): en nuestro caso, existe una probabilidad de cruza de 0,75, si no supera dicho número habrá cruce, en cambio, si supera, no hay cruza y dicho par pasa a la población siguiente sin alterarse. En el caso de haber cruza, se sigue una **cruza de un punto**: se escoge de manera aleatoria un punto de corte en los dos padres (otro número generado al azar de 0 a 29), y se intercambia información genética a la derecha de este punto entre los dos individuos.

8) **Mutación Invertida:** Con una probabilidad de mutación menor a 0.05, generamos un número aleatorio comprendido entre 0 y 1. Si dicho número está por debajo de esa probabilidad, se seleccionará otro número aleatorio entre 0 y 29 correspondiente al gen que se mutará, cambiándose ese gen correspondiente por 0 o 1 según corresponda. Si la probabilidad es mayor, no mutará.

*Los pasos 7 y 8 se repetirán 5 veces para los 5 pares de cromosomas

CONCEPTOS APLICADOS PARA LA RESOLUCIÓN

Para la resolución de este problema, hemos decidido utilizar el ***lenguaje de programación orientado a objetos (POO)***: **JAVA**, por ende hemos utilizado los siguientes conceptos:

- ✓ ***Arreglo:*** Una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo (*int, double, String*). El tamaño de los arrays se declara en un primer momento y no puede cambiar luego durante la ejecución del programa.
- ✓ ***Random:*** La clase Random proporciona un generador de números aleatorios.
- ✓ ***Métodos de la clase String:*** Un conjunto de métodos para el manejo de cadenas string. Por ejemplo, métodos para concatenar cadenas, obtener subcadenas, etc.
- ✓ ***Clase FileWriter:*** La utilizamos para escribir los resultados del programa en archivos externos .txt, para luego generar las gráficas correspondientes en Excel.

Código fuente

```
import java.util.Random;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.math.*;
public class main {

	public static void main(String[] args)
	{

		//Definiciones
		String[] inicial = new String[10];
		int[] numeros = new int[10];
		String[] hijos = new String[10];
		double[] objetivo = new double[10];
		double[] fitness = new double[10];
		double[] promObj = new double[1120];
		double[] maxObj = new double[1120];
		double[] minObj = new double[1120];
		String[] numMax = new String[1120];


		Random rnd = new Random();

		//Coeficiente
		int coef=(int)Math.pow(2, 30) -1;

		//genero cromosoma inicial (agrego el primer digito, y le concateno los otros 29)
		for(int i=0;i<10;i++)
		{
			inicial[i]=Integer.toString(rnd.nextInt(2));
			for(int j=0;j<29;j++)
			{

	inicial[i]=inicial[i].concat(Integer.toString(rnd.nextInt(2)));
			}
		}
		int cantTirada=20;
		int menor =0;
		for(int y=0;y<3;y++)
		{

			if(y==1)
			{
				cantTirada=120;
				menor=20;
			}
			if(y==2)
			{
				cantTirada=1120;
				menor=100;
```

```
                }

            //primer genetico de 20 pasadas
            for(int i=0;i<10;i++)
            {
                  numeros[i]=Integer.parseInt(inicial[i],2);
            }

            for(int z=menor;z<cantTirada;z++)
            {
                  //Obtengo minimo, maximo y suma de Funcion Objetivo
                  double sumaObj = 0;
                  maxObj[z] = 0;
                  minObj[z] = 10;
                  int indiceMax=-1;
                  for(int i=0;i<10;i++)
                  {
                        objetivo[i]= Math.pow((double)numeros[i]/coef, 2);
                        if(objetivo[i]>maxObj[z])
                              {
                              maxObj[z]=objetivo[i];
                              indiceMax=i;
                              }
                        if(objetivo[i]<minObj[z]) minObj[z]=objetivo[i];
                        sumaObj=sumaObj+objetivo[i];
                  }
                  //MANTENEMOS TAMAÑO DE PALABRA DE 30
                  String fix = "000000000000000000000000000000";
                  String number =
Integer.toBinaryString(numeros[indiceMax]);
                  numMax[z] = fix.substring(0,30-
number.length()).concat(number);
                  promObj[z]=sumaObj/10;
                  //Obtengo Funcion Fitness
                  for(int i=0;i<10;i++)
                  {
                        fitness[i]=objetivo[i]/sumaObj;
                  }


                  //RULETA
                  //lo que hacemos es separar por intervalos
consecutivos, y comparar el numero random con el principio y el fin del
intervalo
                  //los intervalos son de 0 al primer fitnes; y despues
van desde el fin del fitnes anterior hasta el fin del fitness concurrente
                  //la suma de todos los fitness es igual a 1
                  int[] elegido = new int[10];
                  for(int j=0;j<10;j++)
                  {
                        double rndRuleta = rnd.nextDouble();
                        double sumfitness = fitness[0];

                        for(int i=0;i<10;i++)
                        {
                              //sumFitness lleva el tope del campo, si le
resto fitness[i] obtengo el inicio del campo
```

```
                              if((rndRuleta>sumfitness) &&
(rndRuleta<(sumfitness+fitness[i])) )
                              {
                                    //guardamos el indice del numero que
salio en un arreglo de indices llamado elegido
                                     elegido[j]=i;
                                     break;
                              } else
                              {
                                    sumfitness = sumfitness+fitness[i];
                              }
                        }
                  }

                  //CrossOver y Mutacion (se hace 5 veces, para los 5
pares) ( por eso aumenta de a 2)
                  for(int h=0;h<10;h=h+2)
                  {
                        //guardamos los cromosomas en un auxiliar
                        String
aux1=Integer.toBinaryString(numeros[elegido[h]]);
                        String
aux2=Integer.toBinaryString(numeros[elegido[h+1]]);
                        //Mantenemos el tamaño de palabra de 30
                        //(al trabajar con numeros en decimal, si el
numero binario empieza con 0, cuando lo pasamos a decimal,
                        //y lo volvemos a pasar a binario, nos queda la
cadena mas corta,
                        //entonces lo que hacemos es agregar tantos 0 al
inicio
                        //como sea necesario para llegar al tamaño de
palabra de 30
                        for (int i=0;i<10;i++)
                        {
                              while(aux1.length() <30)
                              {
                                    String cadena = aux1;
                                    aux1="0".concat(cadena);
                              }
                              while(aux2.length() <30)
                              {
                                    String cadena = aux2;
                                    aux2="0".concat(cadena);
                              }
                        }

                        //generamos numero random y si el numero es menor
que 0.75 hacemos crossOver
                        double rndCrossOver = rnd.nextDouble();
                        if(rndCrossOver <0.75)
                        {
                              int rndPosCross = rnd.nextInt(30);
                              //cambio
                              hijos[h]=aux1.substring(0,
rndPosCross)+aux2.substring(rndPosCross,30);
                              hijos[h+1]=aux2.substring(0,
rndPosCross)+aux1.substring(rndPosCross,30);
```

```
                        }
                        else
                        {
                              hijos[h]=aux1;
                              hijos[h+1]=aux2;
                        }

                        //si el numero random es menor que 0.05 mutamos
                        double rndMut1 = rnd.nextDouble();
                        if(rndMut1<0.05)
                        {
                              int rndPosMut = rnd.nextInt(29);
                              if(rndPosMut== 0)
                              {
                                    if(hijos[h].charAt(rndPosMut) == '0')
                                    {

      hijos[h]="1".concat(hijos[h].substring(1, hijos[h].length()-1));
                                    }
                                    else
                                    {

      hijos[h]="0".concat(hijos[h].substring(1, hijos[h].length()-1));
                                    }
                              }else if( rndPosMut == 29)
                              {
                                    if(hijos[h].charAt(rndPosMut) == '0')
                                    {
                                          hijos[h]=hijos[h].substring(0,
hijos[h].length()-2).concat("1");
                                    }
                                    else
                                    {
                                          hijos[h]=hijos[h].substring(0,
hijos[h].length()-2).concat("0");
                                    }
                              }else
                              {
                                    String primera =
hijos[h].substring(0,(rndPosMut-1));
                                    String segunda =
hijos[h].substring((rndPosMut+1));
                                    if(hijos[h].charAt(rndPosMut) == '0')
                                    {

      hijos[h]=primera.concat("1").concat(segunda);
                                    }
                                    else
                                    {

      hijos[h]=primera.concat("0").concat(segunda);
                                    }
                              }
                        }

                        //si el numero random es menor que 0.05 mutamos
                        double rndMut2 = rnd.nextDouble();
```

```
                        if(rndMut2<0.05)
                        {
                              int rndPosMut = rnd.nextInt(29);
                              if(rndPosMut== 0)
                              {
                                    if(hijos[h+1].charAt(rndPosMut) ==
'0')
                                    {

      hijos[h+1]="1".concat(hijos[h+1].substring(1, hijos[h+1].length()-
1));
                                    }
                                    else
                                    {

      hijos[h+1]="0".concat(hijos[h+1].substring(1, hijos[h+1].length()-
1));
                                    }
                              }else if( rndPosMut == 29)
                              {
                                    if(hijos[h+1].charAt(rndPosMut) ==
'0')
                                    {

      hijos[h+1]=hijos[h+1].substring(0, hijos[h+1].length()-
2).concat("1");
                                    }
                                    else
                                    {

      hijos[h+1]=hijos[h+1].substring(0, hijos[h+1].length()-
2).concat("0");
                                    }
                              }else
                              {
                                    String primera =
hijos[h+1].substring(0,(rndPosMut-1));
                                    String segunda =
hijos[h+1].substring((rndPosMut+1));
                                    if(hijos[h+1].charAt(rndPosMut) ==
'0')
                                    {

      hijos[h+1]=primera.concat("1").concat(segunda);
                                    }
                                    else
                                    {

      hijos[h+1]=primera.concat("0").concat(segunda);
                                    }
                              }
                        }

                  }

                  for(int i=0;i<10;i++)
                  {
```

```
                        numeros[i]=Integer.parseInt(hijos[i],2);

                  }

            }
            //termina primer genetico de 20 repeticiones
      }//fin for 3 repeticiones

            //imprimimos resultados
            System.out.println("   Iteracion       Cromosoma
minF(X)                promF(X)                maxF(X)");
            for(int i=0;i<20;i++)
            {
                  System.out.println("      "+(i+1)+"
"+numMax[i]+"    "+minObj[i]+"    "+promObj[i]+"   "+maxObj[i]);
            }

      System.out.println("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n
");
            System.out.println("   Iteracion       Cromosoma
minF(X)                promF(X)                maxF(X)");
            for(int i=20;i<120;i=i+5)
            {
                  System.out.println("      "+(i+1)+"
"+numMax[i]+"    "+minObj[i]+"    "+promObj[i]+"   "+maxObj[i]);
            }

      System.out.println("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n
");
            System.out.println("   Iteracion       Cromosoma
minF(X)                promF(X)                maxF(X)");
            for(int i=120;i<1120;i=i+50)
            {
                  System.out.println("      "+(i+1)+"
"+numMax[i]+"    "+minObj[i]+"    "+promObj[i]+"   "+maxObj[i]);
            }




//guardamos los numeros en txt para poder generar las graficas en excel
            try {
              BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\minimo20.txt"));
                  for (int i = 0; i < 20; i++) {

                        String numero
="0,".concat((Double.toString(minObj[i])).substring(2,
Double.toString(minObj[i]).length()));
                        out.write(numero + " \n");
                  }
                  out.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\promedio20.txt"));
                for (int i = 0; i < 20; i++) {

                    String numero
="0,".concat((Double.toString(promObj[i])).substring(2,
Double.toString(promObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\maximo20.txt"));
                for (int i = 0; i < 20; i++) {

                    String numero
="0,".concat((Double.toString(maxObj[i])).substring(2,
Double.toString(maxObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\numeros20.txt"));
                for (int i = 0; i < 20; i++) {
                    out.write(numMax[i] + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

          //imprimo para 100
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\minimo100.txt"));
                for (int i = 20; i < 120; i=i+5) {

                    String numero
="0,".concat((Double.toString(minObj[i])).substring(2,
Double.toString(minObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
```

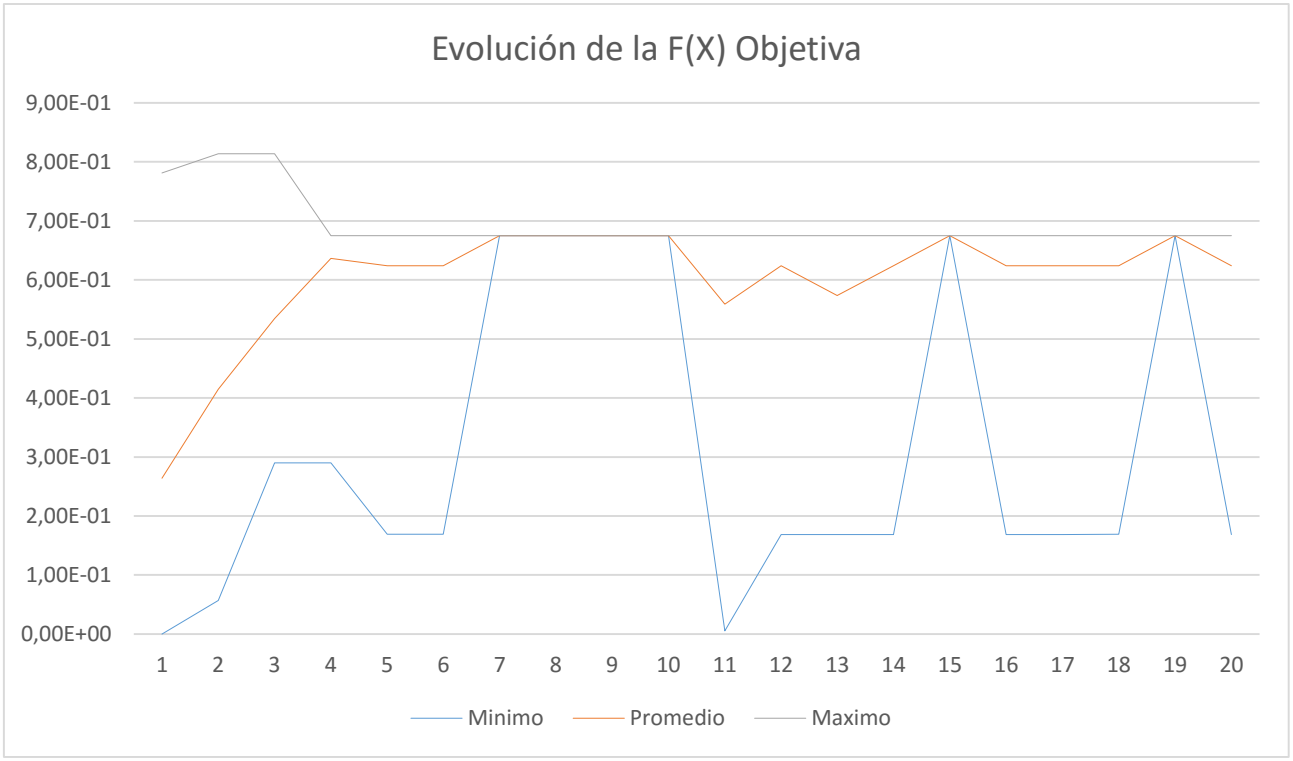
```
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\promedio100.txt"));
                for (int i = 20; i < 120; i=i+5) {

                    String numero
="0,".concat((Double.toString(promObj[i])).substring(2,
Double.toString(promObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\maximo100.txt"));
                for (int i = 20; i < 120; i=i+5) {

                    String numero
="0,".concat((Double.toString(maxObj[i])).substring(2,
Double.toString(maxObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\numeros100.txt"));
                for (int i = 20; i < 120; i=i+5) {
                    out.write(numMax[i] + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

          //imprimo para 1000
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\minimo1000.txt"));
                for (int i = 120; i < 1120; i=i+50) {

                    String numero
="0,".concat((Double.toString(minObj[i])).substring(2,
Double.toString(minObj[i]).length()));
                    out.write(numero + " \n");
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
          try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\promedio1000.txt"));
                for (int i = 120; i < 1120; i=i+50) {
```

```
                        String numero
="0,".concat((Double.toString(promObj[i])).substring(2,
Double.toString(promObj[i]).length()));
                        out.write(numero + " \n");
                    }
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            try {
              BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\maximo1000.txt"));
                    for (int i = 120; i < 1120; i=i+50) {

                        String numero
="0,".concat((Double.toString(maxObj[i])).substring(2,
Double.toString(maxObj[i]).length()));
                        out.write(numero + " \n");
                    }
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            try {
              BufferedWriter out = new BufferedWriter(new
FileWriter("C:\\Users\\nicolas\\desktop\\Geneticos\\numeros1000.txt"));
                    for (int i = 120; i < 1120; i=i+50) {
                          out.write(numMax[i] + " \n");
                    }
                    out.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }


      }//fin main
}//fin clase
```

Resultados

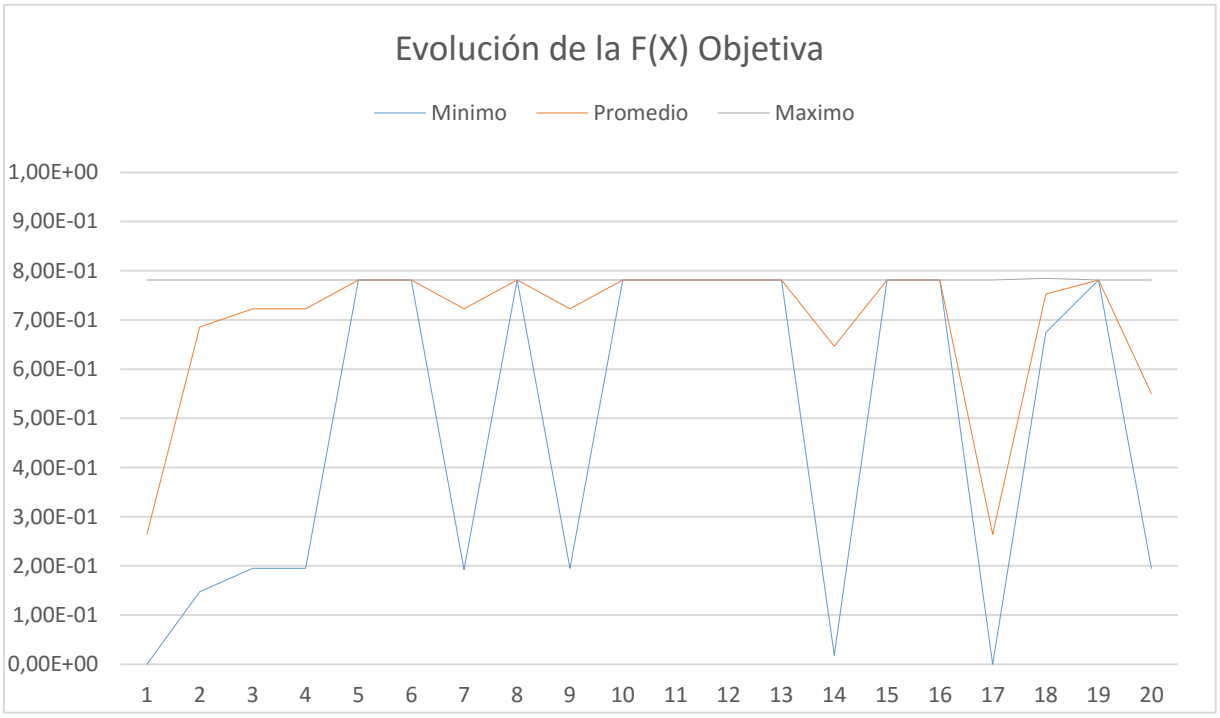
Ciclos del programa: 20

Iteracion	Cromosoma	minF(X)	promF(X)	maxF(X)
1	11100010010000100010001101010	4.3380714785641144E-4	0.26415166431736276	0.781140808313801
2	11100110111010001000110001110	0.057072191202726434	0.4146731866640015	0.8135785863349019
3	11100110111010001000110001110	0.2900322156178301	0.5347283324871569	0.8135785863349019
4	11010010010001111111111001100	0.2900322156178301	0.6362469803001753	0.6747161708052057
5	11010010010001111111111001100	0.1691299508147671	0.6241567838078146	0.6747161708052057
6	11010010010001111111100001100	0.1691299508147671	0.6241566063282036	0.6747158770457417
7	11010010010001111111100001100	0.6747146958885425	0.6747149939317769	0.6747158770457417
8	11010010010001111111110011001100	0.6747146958885425	0.6747149051919779	0.6747149957677335
9	11010010010001111111110011001100	0.6747146958885425	0.6747149051919779	0.6747149957677335
10	11010010010001111111110011001100	0.6747149896477493	0.6747149939317384	0.6747149957677335
11	11010010010001111111110011001100	0.005099440828494067	0.5588395429330795	0.6747149957677335
12	11010010010001111111110011001100	0.16867874894193338	0.6241113680251613	0.6747149957677335
13	11010010010001111111110011100110	0.1686753018784607	0.5735074021552251	0.6747150355476311
14	11010010010001111111110011101000	0.1686768639920751	0.6241111936061394	0.6747150386076234
15	11010010010001111111110011101000	0.6747149957677335	0.6747150040297124	0.6747150386076234
16	11010010010001111111110011101100	0.1686768685820376	0.6241111910051436	0.6747150447276077
17	11010010010001111111110011001100	0.16867843529289045	0.6241110263774237	0.6747149957677335
18	11010010010001111111110011001100	0.1691299508147671	0.624156491272437	0.6747149957677335
19	11010010010001111111110011001100	0.6747149957677335	0.6747149957677336	0.6747149957677335
20	11010010010001111111110011001100	0.1686787443519453	0.6241113706261548	0.6747149957677335



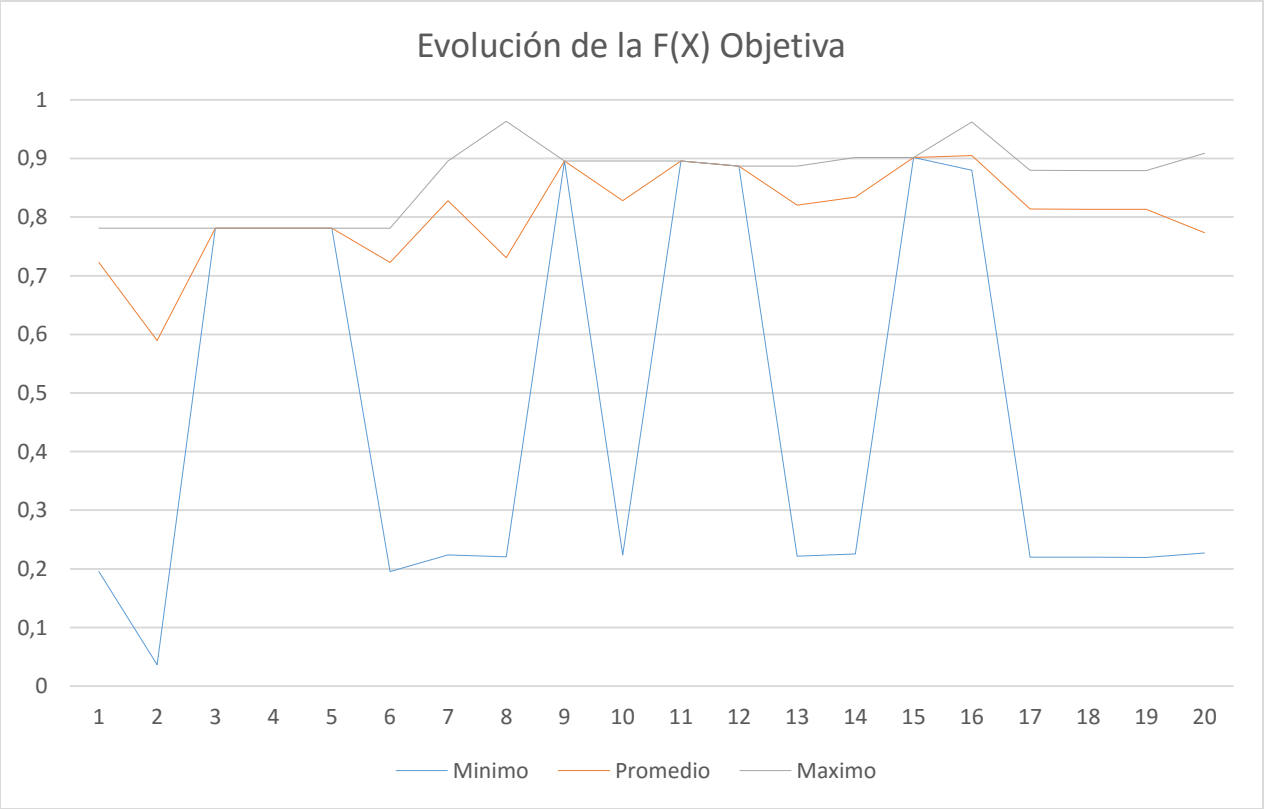
Ciclos del programa: 100

Iteracion	Cromosoma	minF(X)	promF(X)	maxF(X)
21	11100010010000100010011101010	4.3380714785641144E-4	0.26415166431736276	0.781140808313801
26	11100010010000100011001100	0.14738717179851862	0.6858384015884033	0.781142444683396
31	11100010010001111111001100	0.19528443081285984	0.7226523281888846	0.7812975421072575
36	11100010010001111111001100	0.19528558976962634	0.7226808371311081	0.7812975421072575
41	11100010010001111111001100	0.781142444683396	0.7812665226224851	0.7812975421072575
46	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
51	11100010010001111111001100	0.19236809116192588	0.7224045970127243	0.7812975421072575
56	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
61	11100010010001111111001100	0.19532236209248782	0.7227000241057804	0.7812975421072575
66	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
71	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
76	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
81	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
86	11100010010001111111001100	0.017931986993395144	0.6463634685943942	0.7812975421072575
91	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
96	11100010010001111111001100	0.7812975421072575	0.7812975421072574	0.7812975421072575
101	11100010010000100010011101010	4.3380714785641144E-4	0.26415166431736276	0.781140808313801
106	11100010110000100010110101101	0.6745703041500772	0.752839243924255	0.7845980860317829
111	11100010010001111111100011101010	0.781142444683396	0.7812652133968216	0.7812959055752182
116	11100010010001111110110011001001	0.19532067617962115	0.5504747088224979	0.7812948419949527



Ciclos del programa: 1000

Iteracion	Cromosoma	minF(X)	promF(X)	maxF(X)
121	11100010010000100110001100110	0.195302974533807	0.7225630492221077	0.7811475019652521
171	11100010010000100110001100110	0.03683050216770784	0.5897616697376705	0.7811475019652521
221	11100010010000100110001100110	0.7811475019652521	0.7811475019652522	0.7811475019652521
271	11100010010000100110001100110	0.7811472912448479	0.7811474176770904	0.7811475019652521
321	11100010010000100110000100110	0.7811472912448479	0.7811472912448478	0.7811472912448479
371	11100010010000100110000100111	0.19528682116495893	0.7225612445661096	0.781147294537354
421	11110010010000100110000100110	0.22387122959060834	0.8283656647229364	0.8955317130709729
471	11111011010000100110000100110	0.22067715717056938	0.7309924713464093	0.9633061778936256
521	11110010010000100110000100110	0.8955317130709729	0.8955317130709728	0.8955317130709729
571	11110010010000100110000100110	0.22388341300180697	0.8283668830640563	0.8955317130709729
621	11110010010000100110000100110	0.8955262311807353	0.8955311648819491	0.8955317130709729
671	11110001001000100110000100110	0.8872337121126301	0.8872337121126301	0.8872337121126301
721	11110001001000100110000000110	0.22181828657720606	0.8206920685008748	0.8872335998257269
771	11110011001000100110000000110	0.22528847051060577	0.834339920182191	0.9020123034790339
821	11110011001000100110000000110	0.9019543366806313	0.9019833172493771	0.9020123034790339
871	11111011001000000110000011100	0.8798274335413693	0.905167018333802	0.9622823632015925
921	11110000001000000110000011100	0.21995709424994042	0.8138403996122264	0.8798274335413693
971	11110000000100000110000011100	0.21984260821445475	0.8134168015019366	0.8793694896449902
1021	111100000001000000110000011100	0.21972926957384545	0.8134054676378757	0.8793694896449902
1071	111101000001000000110000011100	0.22723093247762674	0.7733435242202833	0.9089182249414346



Conclusiones

Podemos decir que un algoritmo genético es un método de búsqueda dirigida basada en **probabilidad**. Bajo una condición muy débil (que el algoritmo guarde siempre al mejor elemento de la población sin hacerle ningún cambio) se puede demostrar que el algoritmo converge en probabilidad al óptimo. En otras palabras, al aumentar el número de iteraciones, la probabilidad de tener el óptimo en la población tiende a 1 (uno).

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo **se irán incrementando hacia el óptimo global**.

El aspecto central de los AGs es la importancia que cumple el **operador de cruzamiento** y su relación con el **operador de mutación**: mientras el operador cruza tiene una tasa o probabilidad elevada en todo algoritmo genético, el parámetro probabilidad de mutación tiene generalmente valores menos significativos.

Sin embargo, en la resolución de nuestro problema, se agrega una nueva dificultad: la **aleatoriedad**. A la hora tirar la ruleta, probar si hay cruza y si hay mutación, resolvemos estos problemas con la **generación de números aleatorios**.

Es por ello, que durante los ciclos de ejecución podemos notar que ciertos puntos no siguen la regularidad o uniformidad del resto de la población, esto se debe a la aleatoriedad en la generación de los números: pueden cruzar muchas veces seguidas, al igual que mutar**.**

Queda lejos de nuestras manos, hacer que la población sea siempre la misma (se quede con los mejores) y es por ello que surgen picos irregulares en las siguientes gráficas, correspondientes a 20, 100 y 1000 tiradas.

Como podemos observar, en todos los gráficos se intenta tender a un mismo valor, a pesar que aparecen, dependiendo de la cantidad de tiradas del programa que se hagan, algunos picos debido a que los individuos de un población mutaron, y por ende asciende o desciende fuera del rango de regularidad que presenta cada una de las gráficas.

En todos los casos podemos observar que a lo largo de las tiradas, y a pesar de los picos irregulares, la población intenta mejorar, en caso de salir de la “normalidad”, en la siguientes tiradas vuelve a estabilizarse en el valor “normal”. Las mejoras irían incrementándose paulatinamente.

En consecuencia, de la aleatoriedad en la generación de los números, no se puede apreciar en gran medida que las poblaciones hijas mejoran la calidad genética: aumentan (o se mantienen estables) los valores de los promedios, las sumas y los máximos. En cambio, los mínimos presentan mayor variabilidad, por lo que podemos concluir que el efecto que produce dicho comportamiento impredecible de los números para determinar si hay cruza y mutación, afecta en la generación de las poblaciones hijas.

Tanto los máximos como los mínimos de la función objetiva (Fx) para cada población, tienden a ser uno (1) o mantener el mismo valor, es decir, que tienden a tener los mismos individuos en cada población (a pesar de verse alteraciones, debido a la aleatoriedad).