

1er Cuatrimestre 2014

Facultad de Ingeniería

Universidad de Buenos Aires

75.19 – Teoría de la Comunicación

Cátedra Caram – Sarris

Trabajo Práctico – Algoritmos Genéticos

Integrantes

|  |  |  |
| --- | --- | --- |
| Chavar, Hugo | 90541 | hechavar@gmail.com |
| Meller, Diego | 91299 | mellerster@gmail.com |
| Zurita, Stephanie | 91809 | abigail.zurita@gmail.com |

Tabla de contenido

[Introducción a Algoritmos Genéticos 3](#_Toc392798987)

[Descripción del algoritmo genético desarrollado 3](#_Toc392798988)

[Características importantes 3](#_Toc392798989)

[Individuo 3](#_Toc392798990)

[Selección 3](#_Toc392798991)

[Reproducción 3](#_Toc392798992)

[Mutación 3](#_Toc392798993)

[Resultados 3](#_Toc392798994)

[Fuentes 3](#_Toc392798995)

[Uso del Programa 3](#_Toc392798996)

[Anexos 3](#_Toc392798997)

[Salida de la Ejecución 3](#_Toc392798998)

[Código Fuente 3](#_Toc392798999)

# Introducción a Algoritmos Genéticos

Un algoritmo genético es un método adaptativo que usa la búsqueda y el aprendizaje para resolver problemas que de otra manera sería muy difícil. Su objetivo es encontrar una “buena” solución, no garantiza que sea la óptima aunque esta solución se puede mejorar todo lo que se desee aumentando el número de iteraciones o haciendo que la condición de finalización del algoritmo sea más restrictiva.

Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular. Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una Selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

# Descripción del algoritmo genético desarrollado

Para operar, parte de una población inicial y evalúa la aptitud de cada individuo por medio de una función de aptitud que a partir de datos que posee un individuo determina que tan bueno es como solución al problema que se está evaluando. Si no hay una solución satisfactoria, ejecuta un proceso iterativo de selección (elige los individuos más aptos para que su información permanezca en las siguientes generaciones), reproducción (permite que los con más aptitud combinen sus atributos) y mutación (fomenta la diversidad cambiando algunas características al azar); y lo repite las veces necesarias hasta cumplir con la condición de parada.

Un algoritmo genético es un método de búsqueda dirigida basada en probabilidad. Bajo una condición muy débil (que el algoritmo mantenga elitismo, es decir, 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 uno.

Cada individuo representa una solución posible, y se elige la que tenga mejor aptitud.

#### Objetivo del algoritmo

En general los algoritmos genéticos buscan una solución útil a problemas en los que, de otra manera, sería muy difícil y hacerlo de una manera óptima (es decir sin analizar todos los casos posibles). Nuestro desarrollo tiene como finalidad **encontrar el mínimo global de una función continua** , para lo cual se eligió esta función:

Dentro del algoritmo esta función está representada en la *función de aptitud* de cada individuo, y consideraremos una "mejor solución" o "un individuo mejor" al que tiene el menor valor en esta función para cumplir el objetivo de encontrar el mínimo.

Se eligió esta función debido a que su gráfica tiene varios mínimos locales. Estos causan que otros métodos de búsquedas de soluciones globales no sean tan eficientes - midiendo en términos de costo computacional- ni precisos - ya que pueden dar como solución a un mínimo local.

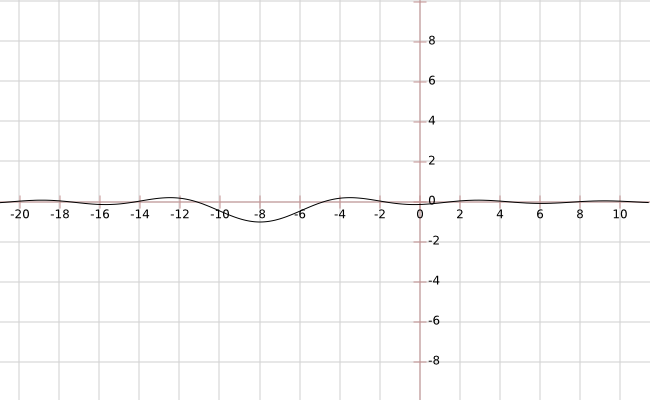


Ilustración : Gráfico de la función de aptitud

Parámetros del algoritmo

A continuación se detallan cuales son los parámetros de entrada del algoritmo, todos estos no varían durante la ejecución del programa.

**Cantidad de individuos en la población**: Cuantos individuos se encuentran, en cada generación, en la población. La población va seleccionando y reproduciendo individuos pero los mismos siempre son esta cantidad. La ventaja de *mantener la cantidad de individuos como constante durante toda la ejecución* hace más fácil la comparación entre generaciones.

**Tasa de selección:** Sirve para saber qué cantidad de individuos mueren debido a la selección, son aquellos que no que tengan menor aptitud. Se mantiene constante durante toda la ejecución del algoritmo, esto significa que en cada generación son alcanzados por la selección la misma cantidad de individuos.

**Tasa De Mutación:** Es la probabilidad de mutación que tienen todos dentro de la población, la cual se mantiene constante durante las generaciones. Se produce que cierta cantidad de estos sufra una mutación, dicha cantidad de individuos no es constante ya que se aplica la evaluación de la tasa por para cada individuo asignándole a este un número al azar y verificando si esta alcanzado por la mutación.

**Condición de fin:** Es una de las condiciones que debe definirse antes de programar el algoritmo, en nuestro caso se determinó que sea la *Mínima Variación Entre Generaciones*, que es la variación entre aptitud poblacional (el promedio entre las aptitudes de todos los individuos que la conforman) en dos iteraciones seguidas.

# Características importantes

## Individuo

Caracterizado :

* **Valor**: que representa los genes del individuo, es un número aleatorio entre -100 y 100.
* **Función de aptitud**: que permita calcular en base al valor asignado de cada individuo, una representación numérica de cuan apto es el mismo. Esta aptitud se utiliza para realizar la selección de los más aptos de la población. Como cuestión de diseño del algoritmo se definió que *es mejor el individuo al que la función otorga el menor valor ya que el objetivo es buscar el mínimo de la función de aptitud*.

## Selección

El operador de selección es el encargado de transmitir y conservar aquellas características de las soluciones que se consideran valiosas a lo largo de las generaciones. El principal medio para que la información útil se transmita es que aquellos individuos mejor adaptados (determinado por la función de aptitud) tengan más probabilidades de sobrevivir.

El algoritmo utiliza la selección por ranking, que consiste en ordenar a los individuos de la población de manera ascendente según la aptitud de los mismos. Según la tasa de selección definida, se eliminan los individuos de menor aptitud, quedando los individuos que poseen mayor aptitud.

## Reproducción

El operador de reproducción permite realizar una exploración de toda la información almacenada hasta el momento en la población y combinarla para crear mejores individuos. Se aplica escogiendo aleatoriamente dos miembros de la población para intercambiar características.

El nuevo individuo debe tener una combinación de los genes (valor de individuo) de los padres, nosotros lo determinamos calculando el promedio de los valores de los progenitores.

## Mutación

Consiste en una alteración de los valores de un cierto número de características de cada individuo, con la intención de fomentar la variabilidad dentro de la población y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual. Esta es la forma de evitar quedarse únicamente con soluciones locales, generando nuevas soluciones en lugares donde no se ha buscado.

El algoritmo utiliza la mutación simple, donde la probabilidad de mutación es constante durante toda la búsqueda. Si la probabilidad de mutación de cada individuo cumple con la probabilidad de mutación constante de la población, entonces el individuo muta y se recalcula la aptitud; caso contrario el individuo no cambia.

# Resultados

Se ejecutó el programa con los siguientes datos:

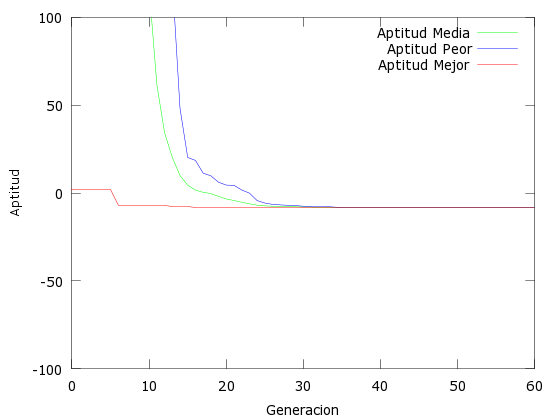
* Cantidad de Individuos en la Población = 10
* Tasa de Selección = 0.1
* Tasa de Mutación = 0.1
* Mínima variación entre Generaciones = 0.0000001

La salida que se obtuvo se puede visualizar en la sección Anexos. En base a la salida que se obtuvo, a continuación se muestra la evolución de la aptitud de la población respecto de las generaciones.

En el ejemplo se puede apreciar que las aptitudes peor y media convergen a la óptima y alrededor de la generación 20 la diferencia entre las mismas es mínima.

# 

Se acorta el intervalo en el eje Y (Aptitud) para poder apreciar la convergencia



# Conclusiones

“Siempre deben existir conclusiones”

Las técnicas de optimización clásicas tienen dificultades al tratar con problemas de optimización global. Una de las principales razones de su fracaso es que pueden quedar fácilmente atrapadas en una solución local (un mínimo local para el problema planteado por este trabajo). Además, son incapaces de generar o usar información global y fallan en los casos de que existen muchas soluciones aproximadamente buenas pero locales y no globales, es decir, no son capaces de hacer una distinción entre soluciones óptimas locales y soluciones óptimas globales, y tratan a las primeras como soluciones actuales del problema original.

El algoritmo genético resuelve problemas de optimización imitando los principios de la evolución biológica, modificando repetidamente una población de soluciones individuales usando reglas de la genética en la reproducción, a fin de evaluar distintas combinaciones como soluciones.

Finalmente, debido a la naturaleza aleatoria en la que busca, un algoritmo genético mejora las oportunidades de encontrar una solución global a un problema de optimización.

# Fuentes

* <http://www.it.uc3m.es/jvillena/irc/practicas/06-07/05.pdf>
* <http://geneura.ugr.es/~jmerelo/ie/ags.htm>
* <http://www.ijcse.com/docs/IJCSE10-01-03-29.pdf>

# Uso del Programa

Para utilizar el programa se debe compilar usando una maquina virtual java. Para ello es necesario tener instalada y configurada una maquina virtual java de desarrollo (JDK) y una maquina virtual java (JVM).

Para modificar los parámetros en el archivo Main.java se puede cambiar las siguientes variables:

int cantidadIndividuosEnPoblacion **=** 100**;**

double tasaDeSeleccion **=** 0.1**;**

double tasaDeMutacion **=** 0.01**;**

double minimaVariacionEntreGeneraciones **=** 0.00000001**;**

Las mismas se encuentran en las líneas 17 a 20 del archivo.

Luego se compila y ejecuta el programa. Se puede hacer por línea de comando, o utilizando el archivo “EjecutarPrograma.bat” que realiza las tareas para poder ejecutar el programa en un entorno Windows.

El programa ejecutado muestra El valor del mejor individuo y un listado de las iteraciones y las aptitudes media, mejor y peor en cada una. Ejemplo de una salida del programa (recortado hasta 20 iteraciones):



Para salir del programa presione cualquier tecla

# Anexos

## Salida de la Ejecución

## 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Generación | Aptitud Media | Aptitud Peor | Aptitud Mejor | Individuos |
| 0 | 2344,91095 | 5380,6982 | 1,81298 | 10 |
| 1 | 2227,17214 | 4537,75449 | 1,81298 | 10 |
| 2 | 2065,28684 | 4203,31009 | 1,81298 | 10 |
| 3 | 1658,2607 | 4203,31009 | 1,81298 | 10 |
| 4 | 1292,94396 | 4155,26414 | 1,81298 | 10 |
| 5 | 878,55393 | 2918,90147 | 1,81298 | 10 |
| 6 | 585,97171 | 2918,90147 | -6,92064 | 10 |
| 7 | 298,21569 | 1043,43921 | -6,92064 | 10 |
| 8 | 285,17225 | 913,00478 | -6,92064 | 10 |
| 9 | 206,1175 | 913,00478 | -6,92064 | 10 |
| 10 | 116,84569 | 550,1427 | -6,92064 | 10 |
| 11 | 61,82902 | 283,38997 | -6,92064 | 10 |
| 12 | 34,4777 | 133,0487 | -6,92064 | 10 |
| 13 | 20,39128 | 122,45729 | -7,81548 | 10 |
| 14 | 10,01258 | 48,31723 | -7,81548 | 10 |
| 15 | 4,53771 | 20,2867 | -7,81548 | 10 |
| 16 | 1,71018 | 18,6703 | -7,9886 | 10 |
| 17 | 0,46272 | 11,38527 | -7,9886 | 10 |
| 18 | -0,23334 | 9,85524 | -7,9886 | 10 |
| 19 | -1,88976 | 6,19571 | -7,9886 | 10 |
| 20 | -3,30511 | 4,5582 | -7,9886 | 10 |
| 21 | -4,18784 | 4,42466 | -7,9886 | 10 |
| 22 | -5,19502 | 1,81298 | -7,9886 | 10 |
| 23 | -6,07994 | -0,02401 | -7,9886 | 10 |
| 24 | -6,87721 | -4,26911 | -7,99662 | 10 |
| 25 | -7,24831 | -5,64719 | -7,99662 | 10 |
| 26 | -7,48246 | -6,43145 | -7,99662 | 10 |
| 27 | -7,62195 | -6,709 | -7,99662 | 10 |
| 28 | -7,70216 | -6,92064 | -7,99662 | 10 |
| 29 | -7,79246 | -7,03622 | -7,99662 | 10 |
| 30 | -7,88644 | -7,51113 | -7,99662 | 10 |
| 31 | -7,93421 | -7,81548 | -7,99662 | 10 |
| 32 | -7,95215 | -7,82363 | -7,99662 | 10 |
| 33 | -7,96976 | -7,82639 | -7,99971 | 10 |
| 34 | -7,98543 | -7,95773 | -7,99971 | 10 |
| 35 | -7,98762 | -7,97599 | -7,99971 | 10 |
| 36 | -7,98923 | -7,97966 | -7,99971 | 10 |
| 37 | -7,99123 | -7,98019 | -7,99971 | 10 |
| 38 | -7,99297 | -7,98304 | -8 | 10 |
| 39 | -7,99462 | -7,9886 | -8 | 10 |
| 40 | -7,99497 | -7,9886 | -8 | 10 |
| 41 | -7,99546 | -7,98886 | -8 | 10 |
| 42 | -7,99645 | -7,99207 | -8 | 10 |
| 43 | -7,99725 | -7,99207 | -8 | 10 |
| 44 | -7,99803 | -7,99357 | -8 | 10 |
| 45 | -7,99867 | -7,99491 | -8 | 10 |
| 46 | -7,99906 | -7,99662 | -8 | 10 |
| 47 | -7,99939 | -7,99723 | -8 | 10 |
| 48 | -7,99967 | -7,99879 | -8 | 10 |
| 49 | -7,99979 | -7,99879 | -8 | 10 |
| 50 | -7,99991 | -7,99958 | -8 | 10 |
| 51 | -7,99994 | -7,99971 | -8 | 10 |
| 52 | -7,99997 | -7,99995 | -8 | 10 |
| 53 | -7,99997 | -7,99995 | -8 | 10 |
| 54 | -7,99998 | -7,99995 | -8 | 10 |
| 55 | -7,99998 | -7,99995 | -8 | 10 |
| 56 | -7,99999 | -7,99995 | -8 | 10 |
| 57 | -7,99999 | -7,99995 | -8 | 10 |
| 58 | -7,99999 | -7,99998 | -8 | 10 |
| 59 | -8 | -7,99999 | -8 | 10 |
| 60 | -8 | -7,99999 | -8 | 10 |

## Código Fuente

Main.java

package genetico**;**

**import** java**.**util**.**ArrayList**;**

public class Main **{**

public static void main**(**String**[]** args**)** **{**

System**.**out**.**println**(**"TP de Redes Complejas / Sistemas Multiagente"**);**

System**.**out**.**println**(**"Alumnos:"**);**

System**.**out**.**println**(**" Stephanie Abigail Zurita"**);**

System**.**out**.**println**(**" Hugo Chavar"**);**

System**.**out**.**println**(**" Diego Meller"**);**

System**.**out**.**println**();**

System**.**out**.**println**();**

// Defino parametros

int cantidadIndividuosEnPoblacion **=** 100**;**

double tasaDeSeleccion **=** 0.1**;**

double tasaDeMutacion **=** 0.01**;**

double minimaVariacionEntreGeneraciones **=** 0.00000001**;**

// Creo la generacion 0 de la poblacion

Poblacion pobl **=** **new** Poblacion**(**cantidadIndividuosEnPoblacion**,**

tasaDeSeleccion**,** tasaDeMutacion**,**

minimaVariacionEntreGeneraciones**);**

ArrayList**<**Double**>** aptitudPoblaciones **=** **new** ArrayList**<**Double**>();**

ArrayList**<**String**>** estAptitudPoblaciones **=** **new** ArrayList**<**String**>();**

**while** **(true)** **{**

pobl**.**seleccion**();**

pobl**.**selectReproduccion**();**

pobl**.**mutacion**();**

aptitudPoblaciones**.**add**(**pobl**.**getAptitudPoblacion**());**

pobl**.**calcAptitudPoblacion**();**

double indMedio **=** pobl**.**getAptitudPoblacion**();**

double indMin **=** pobl**.**getLeastIndividuo**();**

double indMax **=** pobl**.**getBestIndividuo**();**

String res **=** String**.**format**(**"%9s"**,** String**.**format**(**"%.5f"**,** indMedio**))**

**+** "\t "

**+** String**.**format**(**"%9s"**,** String**.**format**(**"%.5f"**,** indMax**))**

**+** "\t "

**+** String**.**format**(**"%9s"**,** String**.**format**(**"%.5f"**,** indMin**));**

estAptitudPoblaciones**.**add**(**res**);**

**if** **(!**pobl**.**esCondicionDeFin**())** **{**

**break;**

**}**

**}**

Individuo mejor **=** pobl**.**getMejor**();**

System**.**out**.**println**(**"El mejor individuo tiene un valor de: "

**+** mejor**.**getValor**());**

System**.**out**.**println**();**

System**.**out

**.**println**(**"Gen.\t Aptitud Media\t Aptitud Peor\t Aptitud Mejor\t Individuo"**);**

int i**;**

**for** **(**i **=** 0**;** i **<** estAptitudPoblaciones**.**size**();** i**++)** **{**

System**.**out**.**println**(**i **+** "\t " **+** estAptitudPoblaciones**.**get**(**i**));**

**}**

**}**

**}**

Poblacion.java

package genetico**;**

**import** java**.**util**.\*;**

public class Poblacion **{**

public int getTamano**()** **{**

**return** tamano**;**

**}**

private ArrayList**<**Individuo**>** listaIndividuos**;**

private int tamano**;**

private double porcSeleccion**;**

private double porcMutacion**;**

private double aptitudPoblacion**;**

private double aptitudAnterior**;**

private double condicionDeFin**;**

// Constructor

Poblacion**(**int tamano**,** double porcSeleccion**,** double porcMutacion**,**

double condDeFin**)** **{**

**this.**tamano **=** tamano**;**

**this.**porcSeleccion **=** porcSeleccion**;**

**this.**porcMutacion **=** porcMutacion**;**

**this.**aptitudPoblacion **=** 0**;**

**this.**condicionDeFin **=** condDeFin**;**

// crear poblacion

listaIndividuos **=** **new** ArrayList**<**Individuo**>();**

int i**;**

**for** **(**i **=** 0**;** i **<** tamano**;** i**++)** **{**

listaIndividuos**.**add**(new** Individuo**(this.**porcMutacion**));**

**}**

**this.**calcAptitudPoblacion**();**

**}**

public double getAptitudPoblacion**()** **{**

**return** **this.**aptitudPoblacion**;**

**}**

public double getCondicionDeFin**()** **{**

**return** **this.**condicionDeFin**;**

**}**

public void setAptitudAnterior**(**double valor**)** **{**

aptitudAnterior **=** valor**;**

**}**

public double getAptitudAnterior**()** **{**

**return** aptitudAnterior**;**

**}**

public void calcAptitudPoblacion**()** **{**

setAptitudAnterior**(this.**getAptitudPoblacion**());**

int i**;**

double result **=** 0**;**

**for** **(**i **=** 0**;** i **<** listaIndividuos**.**size**();** i**++)** **{**

result **=** result **+** listaIndividuos**.**get**(**i**).**getAptitud**();**

**}**

**this.**aptitudPoblacion **=** result **/** listaIndividuos**.**size**();**

**}**

// Selección: eliminar las individuos con la peor aptitud o sea con la aptitud máxima

public void seleccion**()** **{**

// calcular el numero de individuos de borrar

double aux **=** tamano **\*** porcSeleccion**;**

int elim **=** **(**int**)** aux**;**

Collections**.**sort**(**listaIndividuos**);**

Iterator**<**Individuo**>** ite **=** listaIndividuos**.**iterator**();**

// elimino los "elim" elementos máximos de listaIndividuo,

**while** **(**ite**.**hasNext**())** **{**

ite**.**next**();**

ite**.**remove**();**

elim**--;**

**if** **(**elim **==** 0**)** **{**

**break;**

**}**

**}**

**}**

// eligo individuo al azar por la reproduccion

public void selectReproduccion**()** **{**

int noReproduccion **=** tamano **-** listaIndividuos**.**size**();**

int i**;**

**for** **(**i **=** 0**;** i **<** noReproduccion**;** i**++)** **{**

double random1 **=** Math**.**random**()** **\*** **(**listaIndividuos**.**size**()** **-** 1**);**

double random2 **=** Math**.**random**()** **\*** **(**listaIndividuos**.**size**()** **-** 1**);**

int res1 **=** **(**int**)** random1**;**

int res2 **=** **(**int**)** random2**;**

Individuo ind1 **=** listaIndividuos**.**get**(**res1**);**

Individuo ind2 **=** listaIndividuos**.**get**(**res2**);**

Individuo indHijo **=** ind1**.**reproducirCon**(**ind2**);**

listaIndividuos**.**add**(**indHijo**);**

**}**

**}**

// calcular condicion de fin

public boolean esCondicionDeFin**()** **{**

double aptitud **=** **this.**getAptitudPoblacion**();**

double aAnterior **=** getAptitudAnterior**();**

double limit **=** getCondicionDeFin**();**

double check **=** Math**.**abs**(**1 **-** **(**aptitud **/** aAnterior**));**

**return** **(**check **>=** limit**);**

**}**

public void mutacion**()** **{**

int i**;**

**for** **(**i **=** 0**;** i **<** listaIndividuos**.**size**();** i**++)** **{**

listaIndividuos**.**get**(**i**).**mutar**();**

**}**

**}**

public void mostrarIndividuos**()** **{**

Collections**.**sort**(**listaIndividuos**);**

Iterator**<**Individuo**>** ite **=** listaIndividuos**.**iterator**();**

**while** **(**ite**.**hasNext**())** **{**

Individuo s **=** ite**.**next**();**

System**.**out**.**println**(**s**);**

**}**

**}**

public double getBestIndividuo**()** **{**

int i**;**

double max **=** listaIndividuos**.**get**(**0**).**getAptitud**();**

double act**;**

**for** **(**i **=** 1**;** i **<** listaIndividuos**.**size**();** i**++)** **{**

act **=** listaIndividuos**.**get**(**i**).**getAptitud**();**

**if** **(**act **>** max**)** **{**

max **=** act**;**

**}**

**}**

**return** max**;**

**}**

public double getLeastIndividuo**()** **{**

int i**;**

double min **=** listaIndividuos**.**get**(**0**).**getAptitud**();**

double act**;**

**for** **(**i **=** 1**;** i **<** listaIndividuos**.**size**();** i**++)** **{**

act **=** listaIndividuos**.**get**(**i**).**getAptitud**();**

**if** **(**act **<** min**)** **{**

min **=** act**;**

**}**

**}**

**return** min**;**

**}**

public Individuo getMejor**()** **{**

Collections**.**reverse**(**listaIndividuos**);**

Individuo ind **=** listaIndividuos**.**get**(**0**);**

**return** ind**;**

**}**

**}**

Individuo.java

package genetico**;**

**import** java**.**util**.**Random**;**

public class Individuo **implements** Comparable**<**Object**>** **{**

private double aptitud**;**

private float valor**;**

private double probabilidadMutacion**;**

public static Random rand **=** **new** Random**();**

public double getAptitud**()** **{**

**return** aptitud**;**

**}**

public void setValor**(**float valor**)** **{**

**this.**valor **=** valor**;**

**}**

public float getValor**()** **{**

**return** valor**;**

**}**

public Individuo**(**double probabilidadMutacion**)** **{**

**this.**probabilidadMutacion **=** probabilidadMutacion**;**

**this.**valor **=** rand**.**nextFloat**()** **\*** 200 **-** 100**;** // ramdom entre -100 y 100

calcularAptitud**();**

**}**

public Individuo**(**float valor**,** double probabilidadMutacion**)** **{**

**this.**valor **=** valor**;**

**this.**probabilidadMutacion **=** probabilidadMutacion**;**

calcularAptitud**();**

**}**

public void calcularAptitud**()** **{**

aptitud **=** (-1)\*Math**.**pow**(**valor + 8**,** -1**)\***Math**.**sin(valor + 8)**;**

**}**

public void mutar**()** **{**

**if** **(**rand**.**nextDouble**()** **<** **this.**probabilidadMutacion**)** **{**

**this.**realizarMutacion**();**

calcularAptitud**();**

**}**

**}**

public void realizarMutacion**()** **{**

// realizo una intercambio de 3 bits de los 32 que tiene un float

// los la posicion de los bits a cambiar se elije de manera aleatoria

// con una distribucion normal

int valorComoBitsDeEntero **=** Float**.**floatToIntBits**(this.**valor**);**

int aux **=** 0**;**

int cambiosValidos **=** 3**;**

**while** **(**cambiosValidos **>** 0**)** **{**

// los bits mas probables de ser cambiados

// son los centrales

int posicion **=** **(**int**)** **(**3 **\*** rand**.**nextGaussian**()** **+** 15**);**

aux **|=** 1 **<<** posicion**;**

cambiosValidos**--;**

**}**

valorComoBitsDeEntero ^= (aux & 0xffffffff);

this.valor = Float.intBitsToFloat(valorComoBitsDeEntero);

calcularAptitud();

}

@Override

public String toString() {

return "Individuo [ valor=" + String.format("%.5f", this.valor)

+ ",aptitud= " + String.format("%.5f", this.aptitud) + "]";

}

public Individuo reproducirCon(Individuo other) {

Individuo hijo = new Individuo(

(this.getValor() + other.getValor()) / 2,

this.probabilidadMutacion);

return (hijo);

}

@Override

public int compareTo(Object obj) {

final int BEFORE = -1;

final int EQUAL = 0;

final int AFTER = 1;

if (this == obj) {

return EQUAL;

}

if (obj == null) {

return EQUAL;

}

if (!(obj instanceof Individuo)) {

return BEFORE;

}

Individuo other = (Individuo) obj;

if (this.getAptitud() > other.getAptitud())

return BEFORE;

if (this.getAptitud() < other.getAptitud())

return AFTER;

return EQUAL;

}

}