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](#_Toc392440367)

[Descripción del algoritmo genético desarrollado 3](#_Toc392440368)

[Características importantes 4](#_Toc392440369)

[Individuo 4](#_Toc392440370)

[Selección 4](#_Toc392440371)

[Reproducción 4](#_Toc392440372)

[Mutación 5](#_Toc392440373)

[Resultados 5](#_Toc392440374)

[Fuentes 7](#_Toc392440375)

[Uso del Programa 8](#_Toc392440376)

[Código Fuente 9](#_Toc392440377)

# 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), cruzamiento (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.

Parámetros del algoritmo

**Cantidad de individuos en la población**: Cuantos individuos se encuentran en la población original.

**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 tiene cada individuo particular 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 por un valor (que representa los genes del mismo) y una probabilidad de mutar.

* **Valor**: número aleatorio entre -100 y 100.
* **Probabilidad de Mutar**: número aleatorio entre 0.0 y 1.0
* **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*.

## 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 cruce 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

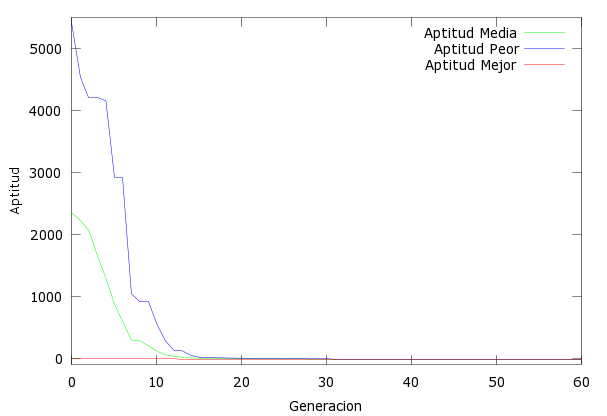
Consiste en una alteración de los valores de un cierto número de características de cada individuo, con la intención de fomentar la variabilidad dentro de la población y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.

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.

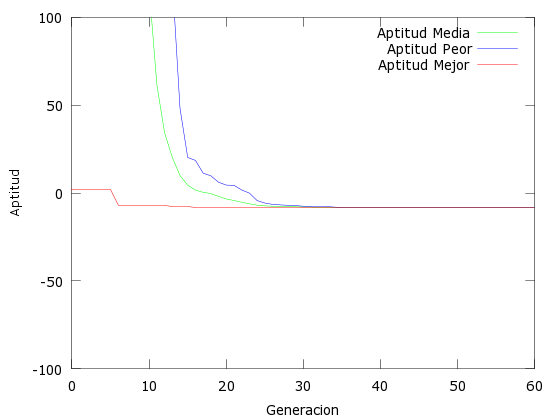
# Resultados

A continuación se muestra la evolución de la aptitud de la población respecto de las generaciones.

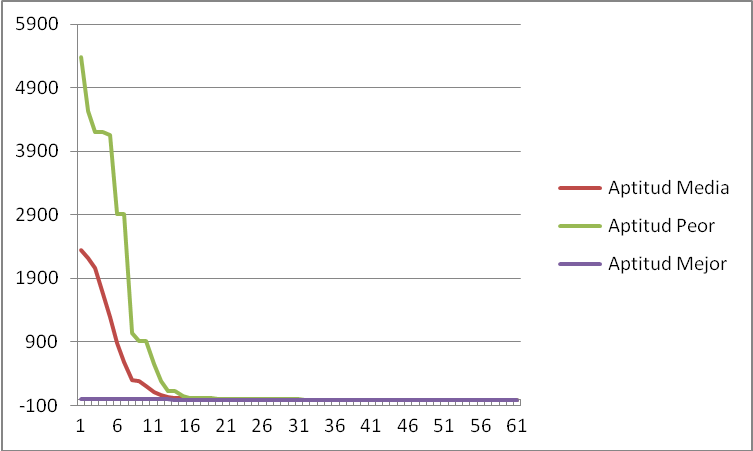
\*Hecho con la herramienta Octave\*



Se acorta el intervalo en y para poder apreciar la convergencia



\*Hecho con Excel\*



# Fuentes

* <http://www.it.uc3m.es/jvillena/irc/practicas/06-07/05.pdf>
* <http://geneura.ugr.es/~jmerelo/ie/ags.htm>

# 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**;** // tasa de reproduccion es el mismo nro

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.

# 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 y veo si hay archivo de configuraci󮍊

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

double tasaDeSeleccion **=** 0.1**;** // tasa de reproduccion es el mismo nro

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**();**

**}**

// Seleccion: eliminar las individuos con la menor aptitud o sea con la

// aptitud maxima

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 maximos 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**()** **{**

**this.**aptitud **=** Math**.**pow**(this.**valor**,** 2**)** **+** 4 **\*** **this.**valor **-** 4**;**

**}**

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;

}

}