Teoría y Aplicaciones de Algoritmos Genéticos

en Ingeniería

Marks, Esteban Darío.
Materia *Automatismos,* UTN FRCU Universidad Tecnológica Nacional
Facultad Regional de Concepción del Uruguay, Entre Ríos, Argentina
E-mail: estebanmarks@hotmail.com

Resumen

Los algoritmos genéticos (de ahora en adelante AG), están inspirados en el principio Darwiniano de la evolución de las especies y su base genética. Los mismos fueron introducidos por John Holland en 1970. Son algoritmos probabilísticos que ofrecen un mecanismo de búsqueda paralela basados en el principio de supervivencia del mas apto y la reproducción.

Los biólogos han estudiado en profundidad la evolución como proceso natural, y aunque quedan aspectos de ése área sin entender, podemos decir que es un campo del conocimiento cuasi completamente explorado.

1. Introducción

John Holland desde pequeño se preguntaba cómo es que la naturaleza creaba seres cada vez mas “perfectos”, y lo elemental de la respuesta vino tiempo después en su adolescencia al comprender el concepto “evolución”. Lo básico de esto, es que se lleva a cabo a través de interacciones locales entre individuos y entre el medio que los rodea.

En los años 50 entró en contacto con las primeras computadoras, donde pudo plasmar algunas ideas que traía desde su niñez: simular el proceso natural de la evolución. A principios de los 60, trabajando en el grupo *Logic of Computers* de la Universidad de Michigan fue donde nacen lo que hoy conocemos como Algoritmos Genéticos.

Allí Holland dictaba el curso *Teoría de sistemas adaptativos* y gracias a la interacción del ambiente intelectual y joven de su alumnado, y a la obra de R. A. Fisher (un biólogo evolucionista), se sentaron las bases teóricas de los AG.

Unos quince años mas adelante, David Goldberg, actual gurú de los AG, conoció a J. Holland, y se convirtió en su estudiante. Éste era un ingeniero industrial trabajando en el diseño de tuberías, y fue el primero que trató de aplicar AG a la resolución de problemas industriales. Holland trató de disuadirlo, argumentando que el problema era muy complicado y que el método por él introducido era apto para problemas mas simples. Pero Goldberg siguió con su trabajo y escribió un algoritmo utilizando una APPLE II el cual resolvió el problema con éxito. En 1985, los AG estaban tan difundidos en la comunidad científica, que se llevó a cabo la primer conferencia ICGA ’85, la cual se sigue realizando bienalmente.

2. Principios

Los AG constituyen una técnica de búsqueda y optimización altamente paralela, inspirados en el principio Darwiniano de la selección natural y la reproducción genética.

Estos principios son muy simples, según Darwin, el principio de selección privilegia a individuos con mayor longevidad y por lo tanto con mayor probabilidad de reproducción. La longevidad es función proporcional con la probabilidad de reproducción. Cuando nos referimos a reproducción, hacemos hincapié en la transferencia de código genético. El código genético nos da la identidad del individuo, y se representan por medio de cromosomas.

Se trata de generar una población, que presente un código genético vinculado con las posibles soluciones de nuestro problema. Y de esta manera generar un sistema que evolucione a soluciones mas aptas.

Es decir, cada cromosoma es sometida al proceso evolutivo: evaluación, selección, recombinación sexual y mutación.

Podemos ver la analogía con la naturaleza en la siguiente tabla

Naturaleza	Algoritmos Genéticos
Cromosoma	Palabra binaria, vector, etc.
Gen	Característica del problema
Alelo	Valor de la característica
Loco	Posición en la palabra
Genotipo	Estructura
Fenotipo	Estructura sometida al problema
Individuo	Solución
Generación	Ciclo

Un algoritmo genético puede ser caracterizado según

1. *Problema a ser optimizado*
2. *Representación de soluciones del problema*
3. *Decodificación del cromosoma*
4. *Evaluación*
5. *Selección*
6. *Operadores genéticos*
7. *Inicialización de la población*
8. *Parámetros y criterios de parada*

2.1. Problema a ser optimizado

La aplicación por excelencia de los GA es la optimización de problemas complejos; los problemas con muchas restricciones, los problemas con muchas variables y una gran interdependencia, problemas altamente no-lineales, problemas donde las demás técnicas de optimización determinista fallan, con grandes espacios de búsqueda, problemas con condiciones que son difíciles de representar matemáticamente.

En el campo de problemas-tipo, fueron aplicados con éxito en la optimización de funciones matemáticas, optimización combinatoria, optimización de planeamiento, problema del viajante, problemas de layout, problemas en PCB electrónico para citar algunos.

2.2. Representación de soluciones del problema

La estructura del cromosoma está definido por las posibles soluciones dentro del espacio de búsqueda del problema. Normalmente se utiliza la representación binaria por ser la mas simple, fácil de manipular y fácil de transformar.

Tomemos como ejemplo la función $F_{(x)}= x^2$

Y *x* es el entero comprendido en el espacio de búsqueda *E* [0,63]

Podemos codificar las diferentes soluciones como un string de 6 bits, si queremos una resolución de 1.

Ejemplo de codificación

C1: 000000	_____	0
C2: 001001	_____	5
Cn: C_i.....C_1	_____	$\Sigma_{(n=1,n=i)}\ C_n.2^{(n-1)}$

Para representar un número en el espacio *E* $[X_{mín},X_{máx}]$ con precisión de *p* cifras decimales, usaremos un string de K bits, según la ecuación

$$2^K<(Xmáx-Xmín).10^p$$

El sistema de representación binario no siempre puede ser utilizado. Muchas veces nuestro problema exige una representación con un alfabeto de diferente cardinalidad, como por ejemplo letras relacionadas con un alfabeto de 26 componentes. Aunque es interesante recalcar que uno de los resultados fundamentales de la teoría de algoritmos genéticos, el *teorema de los esquemas*, afirma que la codificación óptima es la de cardinalidad 2 (binaria).

La mayoría de las veces, el éxito del AG depende de la representación del problema. Generalmente la regla heurística que se utiliza para la codificación es la llamada regla de los *bloques de construcción*, que enuncia “parámetros relacionados entre sí deben estar cercas en el cromosoma”.

Muchas veces, nos encontraremos ante un problema que exija que el cromosoma sea variable en longitud; en éste tipo de problemas, es común usar un cromosoma para que lleve implícito el número de variables. El método de cromosomas variables es el que mas se asemeja a una resolución natural aunque el menos utilizado en la práctica, debido a su complejidad de diseño.

2.3. Decodificación del cromosoma

La decodificación consiste en la construcción de la solución real del problema. En el caso de una codificación binaria como vimos en el ejemplo anterior, la decodificación consiste en convertir la *cadena* binaria en un valor decimal. Aquí podemos apreciar la ventaja de una codificación binaria, ya que la conversión a decimal es muy simple y en muchos lenguajes de programación existen funciones de simple llamada para tal efecto.

2.4. Evaluación

La evaluación es la unión del AG con el mundo externo y las reglas que rigen sobre él. En el proceso de evaluación, se aplica una función sobre nuestra solución para determinar la medida de aptitud *(fitness)* de cada individuo de la población actual. La función es al AG como el medio ambiente lo es a los seres humanos.

La función de evaluación es específica a cada problema; en el ejemplo anterior la función era $F_{(x)}=x^2$.

$F_{(C1=000000)}$ _____ $0^2=0$
$F_{(C2=001001)}$ _____ $5^2=25$
$F_{(Cn=Ci.....C1)}$ _____ $(\Sigma_{(n=1,n=i)}\ C_n.2^{(n-1)})^2$

2.5. Selección

En éste proceso es donde el algoritmo selecciona individuos para la reproducción.

Esta selección se basa en la aptitud del individuo. Individuos mas aptos tienen mas probabilidades de reproducción que individuos menos aptos.

Veamos un ejemplo de cálculo de aptitud de un individuo *i*, sobre una población de *n* individuos.

$$p_i=F(x_i)/\Sigma_{(j=1:j=n)}F(xj)$$

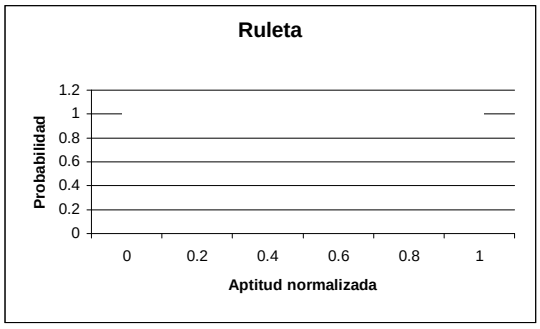
Este tipo de evaluación de aptitud se conoce como *proporcional*.

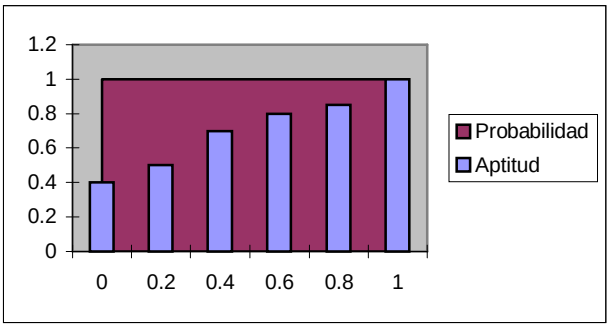
Existen múltiples métodos de escalado aplicables a la aptitud, para que no exista esa proporcionalidad directa. Los mas conocidos son el escalado *sigma* y el *exponencial*. El escalado se aplica cuando la función de aptitud puede dar como resultado que un individuo domine la población. Si no se aplica el escalado, el algoritmo puede converger rápidamente con soluciones locales.

En algunos casos la aptitud o fitness no es solo un número, sinó un vector donde cada componente tiene distinta relevancia. Basta con que dicho fitness sirva para comparar dos individuos y decir cual de ellos es mejor.

Luego de evaluar la aptitud de cada individuo, se procede a la selección propiamente dicha de los individuos. El método mas común es el uso de una "ruleta" donde cada individuo es representado por una porción proporcional de su aptitud.

La gráfica de distribución de probabilidad de una ruleta y la aptitud de cada individuo.





Existen otras formas de seleccionar a los individuos aptos, a saber:

Técnica basada en el Rango: en éste esquema se mantiene un porcentaje de la población, generalmente la mayoría, para la próxima generación. Se ordena la población por aptitud y los *M* individuos menos aptos son eliminados y sustituidos por la descendencia de alguno de los *M* mejores con algún otro individuo de la población.

Selección de Torneo: se escoge aleatoriamente un número *T* de individuos, donde el que tiene mayor puntuación se reproduce, sustituyendo así su descendencia al que tiene menor puntuación.

El *elitismo* es un procedimiento que se combina con los anteriores, y es cuando la generación siguiente siempre incorpora al mejor individuo de la anterior.

2.6. Operadores genéticos

Los individuos seleccionados son reproducidos en la siguiente población, es decir que son recombinados sexualmente para la próxima generación. Al operador genético fundamental de efectuar esta acción se le llama *crossover*.

2.6.1. Crossover n-puntos

Los cromosomas se cortarán en n puntos. Lo mas habitual es 1 o 2 puntos. Aquí vemos un crossover de 2 puntos.

C_1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
C_2	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
$C_1 \Theta C_2$	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
$C_2 \Theta C_1$	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

El punto de corte puede ser o no aleatorio

2.6.2. Crossover uniforme

Se genera un patrón aleatorio de bits y se intercambia la información genética donde hay un 1 en el patrón.

C_1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
C_2	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
patrón	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**
$C_1 \prod C_2$	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
$C_2 \prod C_1$	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**

2.6.3. Crossovers especializados

En algunos problemas es necesario aplicar crossovers que generen siempre soluciones válidas, por ejemplo en el problema del viajante o en problemas de optimización de reticulados. A éste tipo de crossovers se le conoce como especializados.

2.6.4. Mutación

En la Evolución, la mutación es un proceso poco común (1 cada mil replicaciones). En la mayoría de los casos son letales, pero en promedio contribuyen a la diversidad de la población. En un AG tendrán el mismo rol, y necesariamente con la misma probabilidad (<1%). En caso de que la mutación sea muy frecuente, reduce al mecanismo evolutivo del AG en un simple mecanismo aleatorio de búsqueda de soluciones.

La mutación consiste en una vez establecida la frecuencia de mutación (ej. 1/1000), se examina cada bit de cada cadena cuando se vaya a efectuar el crossover o reproducción sexual de sus padres. Si un número generado aleatoriamente está por debajo de esa probabilidad, se procede a invertir el bit. Si no se da la condición anterior, se deja el bit en su estado original. Esto permite que con poblaciones reducidas y cadenas pequeñas, las mutaciones sean raras en una generación. También se pueden encontrar mutaciones "*in natura*", es decir no en el proceso de reproducción sinó en la naturaleza misma (se muta antes o después del crossover, no durante).

C_1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
C_1 mutado	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**

2.6.5. Operador de inversión

Este operador se usa para problemas de *epistasia* (gran interacción entre genes vecinos de un mismo cromosoma).

C_1	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
C_1 **invertido**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**

2.6.6. Otros operadores genéticos

Existen otros operadores genéticos los cuales se usan en la fase final del problema o en problemas donde no se conoce el espacio de búsqueda y el número de variables. Podemos citar a los cromosomas de longitud variable, operadores de nicho (ecológicos) y los operadores especializados como el *zap*, *creep* y *transposition*.

2.7. Inicialización de la población

Designamos como inicialización de la población a la creación de los individuos para el primer ciclo del algoritmo. Normalmente estos individuos son generados aleatoriamente.

Si conocemos aproximaciones buenas a la solución, podemos sembrarlas dentro de una población aleatoria. Por ejemplo, al optimizar la forma de una pieza mecánica podemos sembrar la población con formas-individuos que conocemos que pueden ser soluciones al problema.

2.8. Parámetros y criterios de parada

Parámetros de control del AG

Tamaño de la población: número de puntos del espacio de búsqueda, considerados en paralelo.

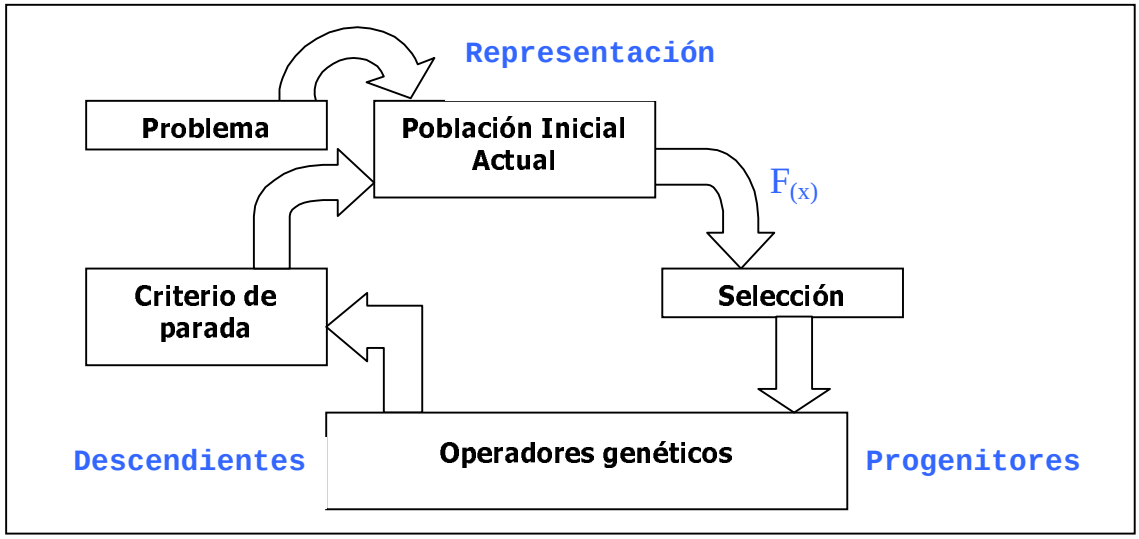
Tasa de crossover: probabilidad de recombinar un individuo con otro.

Tasa de mutación: probabilidad de que una posición/gen sea alterada.

Número de generaciones: número total de ciclos.

Total de individuos: número total de tentativas, es decir $[\Sigma_{(j=1:n)}Individuos]$ (producto del tamaño de la población por el número de generaciones)

Los dos últimos parámetros son utilizados como criterios de parada.



2.9. El zen y los algoritmos genéticos

Este es el título del paper que publicó Goldberg en la ICGA '89. Los consejos para los programadores genéticos son los siguientes:

***Deja que la naturaleza sea tu guía*:** dado que la mayoría de los problemas que serán abordados con AG son de naturaleza no lineal, es mejor actuar como lo hace la naturaleza. *"Si queremos desarrollar sistemas no lineales que busquen y aprendan, es apropiado imitar a sistemas que funcionen"* Goldberg. Estos sistemas se hallan en la naturaleza.

Cuidado con el asalto frontal: el problema ejemplo es la pérdida de diversidad en una población. Hay dos formas de solucionar este problema: aumentando el ritmo de mutación lo que sabemos que puede convertir a nuestro AG en una máquina aleatoria, o bien usando mecanismos como el "sharing" donde la aptitud de un individuo se divide por el número de individuos similares a él.

Respeta la criba de los esquemas: usar alfabetos con baja cardinalidad como el binario.

No te fíes de la autoridad central: La Naturaleza actúa de forma distribuida, tratemos de imitar tal accionar. Por ejemplo, en vez de evaluar la aptitud de un individuo respecto a todos los demás, se puede comparar sólo con los *vecinos*. Es decir, aquellos que de alguna manera estén cerca de él.

3. Aplicaciones en Ingeniería

El límite de los AG reside en la imaginación. En la bibliografía existen incontables ejemplos de diseños exitosos de algoritmos genéticos, por ejemplo en el campo de *diseño de aislaciones acústicas activas* (A. Cirera, UNNE). Otro antecedente exitoso, es el *clustering de redes de telecomunicación* (ISC-MED-35–EGOIST). Dentro del campo de la geología, los algoritmos genéticos ayudan a diseñar *sistemas de control óptimos para estructuras sometidas a sismos* (Arzhang Alimoradi - U. Michigan USA).

Recientemente se desarrolló una aplicación llamada GAGERO, para seleccionar una *unidad de clustering hidráulico para aviones* (V. Kelner – University of Liège). Este es un problema de minimización de peso, costo y tamaño.

En el campo de la ingeniería de imagen, existen prototipos para *optimizar espacios publicitarios* (Castiglione – ORT.uy)

En el campo del Data Mining (extraer conocimiento implícito de datos) se están aplicando los Ags, para *descubrir reglas difusa*s referidas a la información.

Tal vez sea infructuoso detallar las aplicaciones recientes en el campo de los algoritmos genéticos, porque en el mismo momento que Ud. lea estas líneas, seguro que alguien está solucionando problemas usando la evolución como inspiración.

4. Apéndice

GAejemplo.m

```
clear; clc; close all
echo on
% Usamos 6 Bits para un rango de 0 a 63
% La función es un polinomio, ajustado mediante un número de puntos ingresados como ejex y
ejey
% Se usa elitismo, y el miembre ELITE ocupa el lugar Gen(:,1)
% Haciendo algunos cambios como por ejemplo usar un string conjugado de N+N bits para
resolver
% un problema 2-D.
echo off
pause

Gen= (RANDOM('  Discrete Uniform',2,6,4)) - ones(6,4);

%seteo y configuro
ejex=[0
   20
   35
   45
   63];
ejey=[0
   410
   550
   400
   0];
puntotal=size(ejex);
[P,S] = POLYFIT(ejex,ejey,puntotal(1)-1);
X=0:64;
plot(X,P(1)*X.^4 + P(2)*X.^3 + P(3)*X.^2 + P(4)*X + P(5)),xlabel('  Valor de X'),ylabel('  Valor de Y')
vida=100;
% vida = input('  Qué cantidad de generaciones desea? : '  );

GenEliteBin=Gen(:,1);
GenEliteValue=1;
temporal=zeros(vida,1);
% inicio del ciclo
for ciclo=1:vida;


% Calculo el valor en decimal de cada gen
for m=1:4;
   GenValue(m) = (Gen(6,m)* 1) + (Gen(5,m) * 2) + (Gen(4,m) * 4) + (Gen(3,m) * 8) + (Gen(2,m) *
16) + (Gen(1,m) * 32);
end

% Usando la función, encuentro el resultado para cada gen
for m=1:4;
   % Result(m) = (40*GenValue(m)) - (GenValue(m)^2);
   Result(m) = P(1)*GenValue(m)^4 + P(2)*GenValue(m)^3 + P(3)*GenValue(m)^2 +
P(4)*GenValue(m) + P(5);
      if Result(m) >= GenEliteValue
      GenEliteValue=Result(m);
            GenEliteBin = Gen(:,m);
      GenElite = m;
```

```
   else
      end
end

% Resultado total...
Resultotal = Result(1) + Result(2) + Result(3) + Result(4);

% Fitness de cada gen
for m=1:4;
Peso(m)= Result(m) * 1000 / Resultotal;
end

% Reproducción, reproducir cada string in la nueva población proporcionalmente a su contrubución
en la suma
for m=1:4;
r = RANDOM('  Discrete Uniform',1000,1,1);
if r <= Peso(1)
   for i=1:6
   Gen(i,m)=Gen(i,1);
      end
elseif r <= Peso(1)+Peso(2)
   for i=1:6
   Gen(i,m)=Gen(i,2);
      end
elseif r <= Peso(1)+Peso(2)+Peso(3)
   for i=1:6
   Gen(i,m)=Gen(i,3);
      end
else
   for i=1:6
   Gen(i,m)=Gen(i,4);
      end
end
end

%Elitismo
Gen(:,1)=GenEliteBin;

% Crossover
puntocruce1 = RANDOM('  Discrete Uniform',6,1,1);
% Gen 1
% Backup
for m=1:4;
for i=1:6;
   GenBackUp(i,m)=Gen(i,m);
end
end

%Crossover para gen 1
for i=1:6;
   if i<=puntocruce1
      Gen(i,1) = GenBackUp(i,2);
   else
   end
end
```

```
%Crossover para gen 2
for i=1:6;
   if i<=puntocruce1
      Gen(i,2) = GenBackUp(i,1);
   else
   end
end


puntocruce2 = RANDOM('  Discrete Uniform',6,1,1);

%Crossover para gen 3
for i=1:6;
   if i<=puntocruce2;
      Gen(i,3) = GenBackUp(i,4);
   else
   end
end

%Crossover para gen 4
for i=1:6;
   if i<=puntocruce2;
      Gen(i,4) = GenBackUp(i,3);
   else
   end
end

% Elitismo
Gen(:,1)=GenEliteBin;

% Mutación
 puntomutacion = RANDOM('  Discrete Uniform',6,1,1);
 genmutado = RANDOM('  Discrete Uniform',3,1,1);
 Gen(puntomutacion,genmutado+1)=1-Gen(puntomutacion,genmutado+1);

% fin del ciclo
%GenValue(1),GenValue(2),GenValue(3),GenValue(4)
% verificar = input('  Presione ENTER para seguir iterando'  );
temporal(ciclo)=GenValue(1);
end

lll=1:1:vida;
figure,plot(lll,temporal(lll)),xlabel('  Generaciones'  ),ylabel('  X Máximo Maximorum ó Elite')
% figure,plot(lll,P(1)*temporal(lll).^4 + P(2)*temporal(lll).^3 + P(3)*temporal(lll).^2 + P(4)*temporal(lll)
+ P(5)),xlabel('  Generaciones'  ),ylabel('  Y Máximo Maximorum ó Elite'  )
```

5. Bibliografía

- **Illinois Genetic Algorithms Laboratory Website http://www-illigal.ge.uiuc.edu/index.php3**
- **D. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning" Addison-Wesley 1989**
- **Z. Michalewicz "Genetic Algorithms + Data Structures = Evolution Programs" Verlag 1994.**
- **L. Davis "Genetic Algorithms Handbook" VNR Comp. Library 1990.**
- **D. Goldberg, "Zen and genetic algorithms" ICGA '89**