



Programación Evolutiva

Tema 3: Implementación del algoritmo genético simple

Carlos Cervigón, Lourdes Araujo 2023-2024

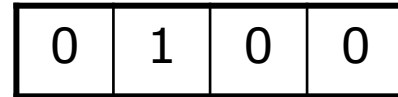


Seudocódigo

```
t=0;  
Generar poblacion inicial(P(t));  
Evaluar población(P(t));  
mientras (t<Num_max_gen) y no CondTermina() {  
    t++;  
    Poblacion(t) = Selección(P(t-1));  
    Reproducción(P(t));;  
    Mutacion(P(t));  
    Evaluar población(P(t));  
}
```

Algoritmo genético simple

- El AGS (Goldberg, 1989) es un algoritmo genético que incorpora los siguientes métodos y criterios:
 - **Criterio de codificación**: Específico de cada problema. Debe hacer corresponder a cada punto del dominio del problema un elemento del espacio de búsqueda: cadenas binarias.



4

genotipo

fenotipo

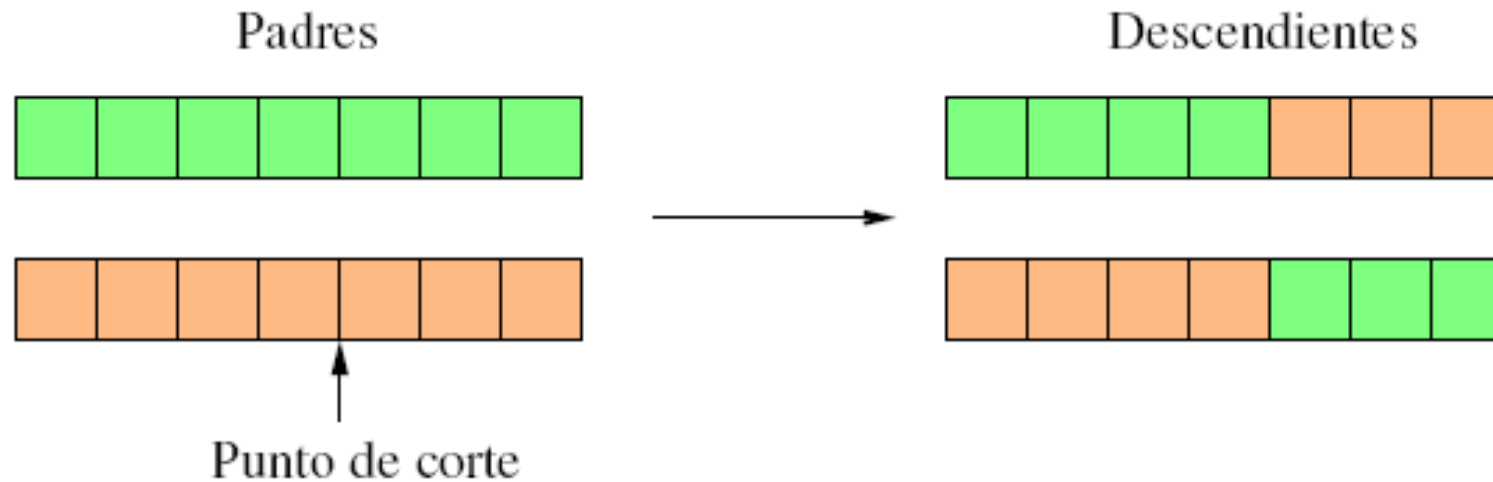
- **Criterio de tratamiento de los individuos no factibles**: No hay. Se considera que la codificación se hace de tal manera que todas las cadenas posibles representan a individuos factibles.
- **Criterio de inicialización**: La población inicial está formada por **cadenas binarias** generadas al azar.

Algoritmo genético simple

- ❑ **Funciones de evaluación y aptitud:** La función de aptitud coincide con la de evaluación. La función de evaluación viene dada a través de una función objetivo.
- ❑ **Operadores genéticos:** Cruce monopunto y mutación bit a bit sobre los individuos codificados.
- ❑ **Criterio de selección:** Ruleta, estocástico universal...
- ❑ **Criterio de reemplazo:** Inmediato.
- ❑ **Criterio de parada:** Fijando el número máximo de iteraciones.
- ❑ **Parámetros de funcionamiento:** Discrecionales. Una posible elección
 - Tamaño población 100
 - Número de generaciones 100
 - Probabilidad de cruce 60%
 - Probabilidad de mutación 2%

Cruce monopunto

- El cruce monopunto genera dos descendientes a partir de dos progenitores cortándolos en una posición elegida al azar e intercambiando los respectivos segmentos.



¿Qué individuos de la población se se cruzan?

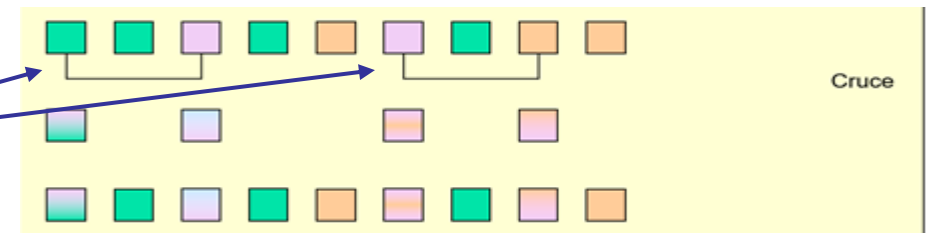
- La determinación de las parejas a cruzar y a mutar se realiza de acuerdo con las probabilidad de aplicación de los operadores genéticos.
 - Para cada individuo se genera un número aleatorio que se compara con la probabilidad de cruce P_{cruce}

$$r_i \leftarrow \text{alea } [0,1]$$

- Se seleccionan para el cruce todos los individuos V_i para los que

$$r_i < P_{cruce}$$

Los individuos seleccionados se van cruzando de dos en dos.



Mutación

- La mutación es el operador básico de alteración.
- Se aplica a individuos solos, realizando una pequeña modificación en alguno de sus genes o en el conjunto.



- Para la mutación se realiza el mismo proceso con cada uno de los $(n \times l)$ bits de la población y usando la probabilidad de mutación P_{mut}

Ejemplo 1: maximización de una función

$$f(x) = \text{abs}((x-5)/(2+\text{sen}(x)))$$

$$x \in [0,15]$$

<i>Representación:</i>	<i>cadena binaria de 4 bits</i>
<i>Selección:</i>	<i>ruleta</i>
<i>Prob. Cruce:</i>	<i>0.7</i>
<i>Prob. Mutación:</i>	<i>0.3</i>
<i>Tamaño población:</i>	<i>2</i>
<i>Número de generaciones:</i>	<i>4</i>

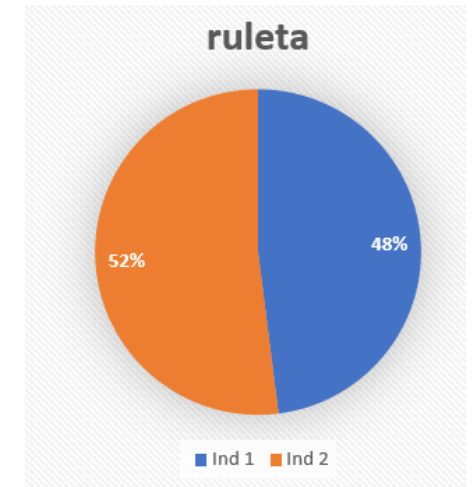
individuos

0111
0100

Ejemplo 1: maximización de una función

GENERACIÓN 1

	genotipo	fenotipo x	$f(x)$	$p.selección$	$p. acum$
Ind 1	0111	7	0.75	0.48	0.48
Ind 2	0100	4	0.80	0.52	1.00
			1.55		

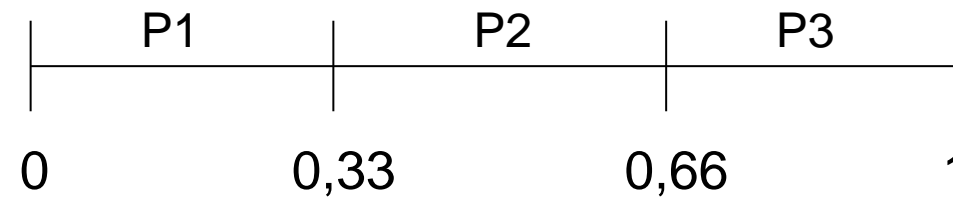


- **SELECCIÓN** -> generamos un número aleatorio entre 0 y 1. En este caso, según la tabla de calidades:
 - Si el número aleatorio cae en el intervalo $[0, 0.48)$ el individuo elegido será el 0111.
 - Si el valor aleatorio cae dentro del intervalo $[0.48, 1)$ elegiremos el individuo 0100. (ej: 0111 y 0100)
- **CRUCE** : Ejecutar el cruce con probabilidad P_{cruce} . Si no hay que emparejar, salir.

$0,15 < 0,7$ -> si hay cruce

Ejemplo 1: maximización de una función

- Elegir un punto de corte de las cadenas entre 1 y L-1. Miro el punto de corte con una elección equiprobable entre los 3 posibles puntos ($1/N$).



0,85 -> como cae en el tercer intervalo, el punto de corte es 3

- Las cadenas que representan a los individuos se parten en dos trozos (por el punto de corte) y se intercambian, dando lugar a dos individuos nuevos:

011|1 **010|0**

- Por tanto los 2 nuevos individuos son:

0110 y **0101**

Ejemplo 1: maximización de una función

■ **MUTACIÓN**

- Para cada gen miro si hay que mutar generando un número aleatorio:
 - Si el número aleatorio es menor que la *ProbMutación* (0,3) entonces cambia el gen por su complementario
 - Si no se deja el gen como está.

```
for (int i=0; i<bits.length; i++) {  
    if (r.nextDouble() < tasaMutacion) {  
        bits[i] = r.nextBoolean();  
        cambios = true;  
    }  
}
```

0110	0.51→0	0.44→1	0.89→1	0.85→0	0110
0101	0.43→0	0.07→0	0.97→0	0.93→1	0001

Ejemplo 1: maximización de una función

GENERACIÓN 2

	genotipo	Fenotipo x	$f(x)$	p.selección	p. acum
Ind 1	0110	6	0.58	0.29	0.29
Ind 2	0001	1	1.41	0.71	1.00
			1.99		

- ❑ **SELECCIÓN** -> Generamos un número aleatorio entre 0 y 1. En este caso, según la tabla de calidades:
 - Si el número aleatorio cae en el intervalo $[0,0.29]$ el individuo elegido será el 0110.
 - Si el valor aleatorio cae dentro del intervalo $[0.29,1)$ elegiremos el individuo 0001. (ej: 0110 y 0001)
- ❑ **CRUCE** → Ejecutar el cruce con probabilidad P_{cruce} . Si no hay que emparejar, salir.
0,75 > 0,7 -> no hay cruce

Ejemplo 1: maximización de una función

■ **MUTACIÓN**

- Para cada gen miro si hay que mutar generando un número aleatorio:
 - Si el número aleatorio es menor que la P. Mutación (0,3) entonces cambia el gen por su complementario.
 - Si no se deja el gen como está.

0110	0.90→0	0.51→1	0.62→1	0.67→0	0110
0001	0.15→1	0.89→0	0.87→0	0.86→1	1001

Ejemplo 1: maximización de una función

GENERACIÓN 3

	genotipo	Fenotipo x	f(x)	p.selección	p. acum
Ind 1	0110	6	0.58	0.26	0.26
Ind 2	1001	9	1.65	0.74	1.00
			2.23		

- ❑ **SELECCIÓN** -> Generamos un número aleatorio entre 0 y 1. En este caso, según la tabla de calidades:
 - Si el número aleatorio cae en el intervalo $[0, 0.26)$ el individuo elegido será el 0110.
 - Si el valor aleatorio cae dentro del intervalo $[0.26, 1)$ elegiremos el individuo 1001.(ej: 1001 y 1001)
- ❑ **CRUCE** → Ejecutar el cruce con probabilidad P_{cruce} . Si no hay que emparejar, salir.
 $0,84 > 0,7$ -> no hay cruce

Ejemplo 1: maximización de una función

■ **MUTACIÓN**

- Para cada gen miro si hay que mutar generando un número aleatorio:
 - Si el número aleatorio es menor que la P. Mutación (0,3) entontes cambiar el gen por su complementario (con alfabeto no binario para elegir un elemento del alfabeto se utiliza el procedimiento de elección equiprobable).
 - Si no se deja el gen como está.

1001	0.98→1	0.33→0	0.25→1	0.07→0	1010
1001	0.18→0	0.84→0	0.27→1	0.08→0	0010

Ejemplo 1: maximización de una función

GENERACIÓN 4

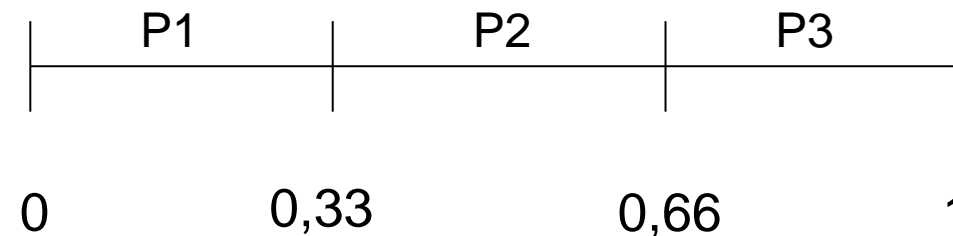
	genotipo	Fenotipo x	f(x)	p.selección	p. acum
Ind 1	1010	10	3.43	0.77	0.77
Ind 2	0010	2	1.03	0.23	1.00
			4.46		

- ❑ **SELECCIÓN** -> Generamos un número aleatorio entre 0 y 1. En este caso, según la tabla de calidades:
 - Si el número aleatorio cae en el intervalo $[0,0.77)$ el individuo elegido será el 1010.
 - Si el valor aleatorio cae dentro del intervalo $[0.77,1)$ elegiremos el individuo 0010.(ej:1010 y 1010)
- ❑ **CRUCE** → Ejecutar el cruce con probabilidad P_{cruce} . Si no hay que emparejar, salir.

$0,28 < 0,7 \rightarrow$ si hay cruce

Ejemplo 1: maximización de una función

- Elegir un punto de corte de las cadenas entre 1 y L-1. Miro el punto de corte con una elección equiprobable entre los 3 posibles puntos.



0,6 -> como cae en el tercer intervalo, el punto de corte es 2

- Las cadenas que representan a los individuos se parten en dos trozos (por el punto de corte) y se intercambian, dando lugar a dos individuos nuevos:

10|10

10|10

- Por tanto los 2 nuevos individuos son:

1010

y

1010

Ejemplo 1: maximización de una función

■ **MUTACIÓN**

- Para cada gen miro si hay que mutar generando un número aleatorio:
 - Si el número aleatorio es menor que la P. Mutación (0,3) entonces cambia el gen por su complementario (con alfabeto no binario para elegir un elemento del alfabeto utilizo el procedimiento de elección equiprobable).
 - Si no se deja el gen como está.
 - Después de 4 iteraciones el individuo de mayor calidad es :

1010

Ejemplo 2: maximización de una función

- ❑ Vamos a aplicar nuestro algoritmo genético a la búsqueda del máximo en el intervalo $[0,20]$ de una sencilla función:

$$f(x) = \frac{x}{1+x^2}$$

- ❑ La sencillez de la función no justifica la necesidad de aplicación de un AG. Sin embargo, precisamente esta sencillez nos permite utilizarla para clarificar el funcionamiento del AG, que se aplicaría exactamente de la misma forma a otras funciones más complejas.
- ❑ Hemos elegido los siguientes valores para los parámetros:
 - PREC : Precisión de la representación 0.0001
 - Tamaño de población 30
 - Probabilidad de cruce 40%
 - Probabilidad de mutación 1%
 - Número de generaciones 10

Ejemplo 2: maximización de una función

- ❑ Los individuos, cromosomas o soluciones serán los puntos del espacio de búsqueda, es decir valores de x en el intervalo $[0..20]$
- ❑ Los valores los codificamos utilizando 0 y 1

$X = 0.4985 \longrightarrow$

1	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---

$X = 12.4577 \longrightarrow$

1	0	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---

...

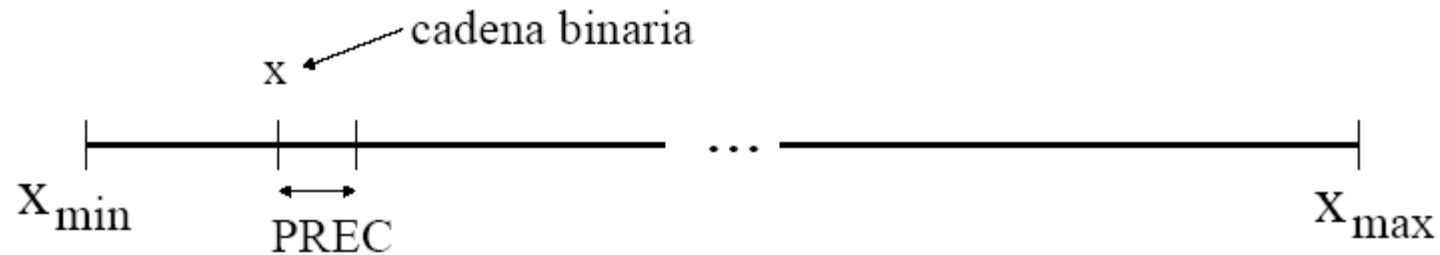
- ❑ Buscamos el valor x que maximiza la función
- ❑ Representación de los individuos o soluciones mediante cadenas binarias

1	0	1	0	1	1	0	0	1
---	---	---	---	---	---	---	---	---

- ❑ ¿Cuál será la longitud del cromosoma?
- ❑ Debo codificar con cadenas de bits los posibles valores del intervalo con la precisión apropiada.

Ejemplo 2: maximización de una función

- ❑ Determinación de la longitud del cromosoma
- ❑ Para discretizar el intervalo real contenido entre x_{\min} y x_{\max} , lo dividimos en pequeñas porciones de anchura menor que PREC, como muestra la figura.



$$\text{tamaño porción} = \frac{x_{\max} - x_{\min}}{2^l - 1} < \text{PREC}$$

- ❑ Siendo l la longitud del cromosoma o individuo

Ejemplo 2: maximización de una función

- De esta fórmula podemos obtener la longitud del cromosoma o cadena binaria que necesitamos utilizar para que nuestro algoritmo pueda alcanzar la precisión requerida:

$$l = \left\lceil \log_2 \left(1 + \frac{x_{max} - x_{min}}{PREC} \right) \right\rceil$$

- En nuestro caso, en que $x_{min} = 0$ y $x_{max} = 20$, tenemos

$$l = \left\lceil \log_2 \left(1 + \frac{20 - 0}{0,0001} \right) \right\rceil = 18$$

00000000000000000000 se corresponde con el valor 0
...
11111111111111111111 se corresponde con el valor 20

De cada individuo tenemos:

- Su **genotipo** (la cadena binaria) :

100000010100101011

- Su **fenotipo** (el valor real que le corresponde a la cadena binaria):

$x = 0.709231$


- La adaptación o **fitness**: en este caso $f(x) \rightarrow f(0.709231)=0.471874$

$f(x) = 0.471874$


$$x(\mathbf{v}) = x_{min} + \text{bin2dec}(\mathbf{v}) \cdot \frac{x_{max} - x_{min}}{2^{l_{crom}} - 1}$$

Ejemplo 2: maximización de una función

- ❑ Una vez calculada la longitud de las cadenas binarias de la población podemos generar la población inicial de forma aleatoria. La siguiente tabla muestra el resultado de esta operación.
- ❑ La tabla presenta para cada individuo:
 - Su **genotipo** (la cadena binaria v) **100000010100101011**
 - Su **fenotipo** (el valor real que le corresponde a la cadena binaria) **0.709231**
 - La adaptación o **fitness**: **0.471874**
 - ¿Cómo obtengo el fenotipo partiendo del genotipo?


$$x(\mathbf{v}) = x_{min} + \text{bin2dec}(\mathbf{v}) \cdot \frac{x_{max} - x_{min}}{2^{l_{crom}} - 1}$$

Ejemplo 2: maximización de una función

Posición	Individuo	x	adaptación
1	000010100010010000	0.709231	0.471874
2	100000010100101011	16.6115	0.059982
3	101110010101101001	11.7698	0.084354
4	001001101111001100	4.05061	0.232694
5	000011011110111101	14.8377	0.0670911
6	001010110011111001	12.4381	0.0798818
7	10000000110111000	2.30477	0.365143
8	110001101101010100	3.34741	0.274262
9	000000110000111110	9.70219	0.101986
10	100010000000101000	1.5638	0.453871
11	000000011011001011	16.5137	0.0603344
12	100111011110100111	17.9634	0.0554968
13	001000111101111111	19.9174	0.0500812
14	010010110110110011	16.0708	0.0619848
15	101111001000110010	5.96171	0.163147
16	001100101101101011	16.7832	0.0593726
17	100101100001010000	0.789264	0.486318
18	010011100010010101	13.2119	0.0752583
19	001000011011000100	2.76399	0.31992
20	110101110001101010	6.7367	0.14524

Ejemplo 2: maximización de una función

- ❑ En este caso la adaptación media de la población es de 0.183415 y el mejor individuo es el de la posición 17, con un valor de adaptación de 0.486318.
- ❑ Podemos observar que en la población inicial los valores de las adaptaciones son muy variados.
- ❑ La siguiente fase es el proceso de **selección** de supervivientes por el método de la ruleta.
- ❑ La siguiente tabla muestra en la primera columna los valores generados por este método y la población resultante del proceso.

Ejemplo 2: maximización de una función

Selección	Individuo	x	adaptación
11	000000011011001011	16.5137	0.0603344
10	100010000000101000	1.5638	0.453871
19	001000011011000100	2.76399	0.31992
6	001010110011111001	12.4381	0.0798818
15	101111001000110010	5.96171	0.163147
17	100101100001010000	0.789264	0.486318
20	110101110001101010	6.7367	0.14524
3	101110010101101001	11.7698	0.084354
17	100101100001010000	0.789264	0.486318
8	110001101101010100	3.34741	0.274262
20	110101110001101010	6.7367	0.14524
20	110101110001101010	6.7367	0.14524
8	110001101101010100	3.34741	0.274262
12	100111011110100111	17.9634	0.0554968
17	100101100001010000	0.789264	0.486318
17	100101100001010000	0.789264	0.486318
8	110001101101010100	3.34741	0.274262
2	100000010100101011	16.6115	0.059982
17	100101100001010000	0.789264	0.486318
6	001010110011111001	12.4381	0.0798818

Ejemplo 2: maximización de una función

- ❑ Tras el proceso de selección, la adaptación media de la población sube a 0.252348.
- ❑ Los individuos más adaptados, como el 17, tienden a recibir más copias en la nueva población
- ❑ Los individuos de baja adaptación, como el 2, tienden a desaparecer.

Ejemplo 2: maximización de una función

- ❑ El siguiente paso de la evolución es la aplicación del operador de **cruce**.
- ❑ La siguiente tabla muestra los padres elegidos aleatoriamente para el cruce comprobando la tasa de mutación, y el punto de cruce, también elegido aleatoriamente.

Punto de cruce	padre1	padre2
13	2	3
4	7	12
3	13	15
3	16	17
16	18	19

Y después del cruce:

Ejemplo 2: maximización de una función

Posición	Individuo	x	adaptación
1	000000011011001011	16.5137	0.0603344
2	100010000000100100	2.81381	0.315537
3	001000011011001000	1.51398	0.459877
4	001010110011111001	12.4381	0.0798818
5	101111001000110010	5.96171	0.163147
6	100101100001010000	0.789264	0.486318
7	110101110001101010	6.7367	0.14524
8	101110010101101001	11.7698	0.084354
9	100101100001010000	0.789264	0.486318
10	110001101101010100	3.34741	0.274262
11	110101110001101010	6.7367	0.14524
12	110101110001101010	6.7367	0.14524
13	110101100001010000	0.789416	0.48634
14	100111011110100111	17.9634	0.0554968
15	100001101101010100	3.34726	0.274272
16	100001101101010100	3.34726	0.274272
17	110101100001010000	0.789416	0.48634
18	100000010100101000	1.61141	0.448032
19	100101100001010011	15.7893	0.0630809
20	001010110011111001	12.4381	0.0798818

Ejemplo 2: maximización de una función

- ❑ En este caso, la adaptación media ha descendido ligeramente, a 0.250673.
- ❑ El mejor individuo en este caso sigue siendo el de la posición 17, y es el producto de uno de los cruces, teniendo un valor de adaptación ligeramente mejor (0.48634) que el de sus progenitores (0.486318 y 0.274262).
- ❑ Finalmente se aplica el operador de mutación. La siguiente tabla muestra los posiciones de los individuos y posiciones (dentro del individuo) de los genes que han mutado.

Individuo	gen
9	15
10	16
14	8
20	2

Ejemplo 2: maximización de una función

Posición	Individuo	x	adaptación
1	000000011011001011	16.5137	0.0603344
2	100010000000100100	2.81381	0.315537
3	001000011011001000	1.51398	0.459877
4	001010110011111001	12.4381	0.0798818
5	101111001000110010	5.96171	0.163147
6	100101100001010000	0.789264	0.486318
7	110101110001101010	6.7367	0.14524
8	101110010101101001	11.7698	0.084354
9	100101100001011000	2.03927	0.395313
10	110001101101010000	0.8474	0.493223
11	110101110001101010	6.7367	0.14524
12	110101110001101010	6.7367	0.14524
13	110101100001010000	0.789416	0.48634
14	100111001110100111	17.9536	0.0555268
15	100001101101010100	3.34726	0.274272
16	100001101101010100	3.34726	0.274272
17	110101100001010000	0.789416	0.48634
18	100000010100101000	1.61141	0.448032
19	100101100001010011	15.7893	0.0630809
20	011010110011111001	12.4382	0.0798808

Ejemplo 2: maximización de una función

- Tras la mutación, la adaptación media de la población ha pasado a ser 0.257073. El mejor individuo es ahora el 10, es el producto de una mutación, con una adaptación de 0.493223, un valor que ya se acerca al óptimo de la función que está en 0.5.
- La evolución de los valores medio y máximo de la adaptación con las generaciones aparece en la siguiente tabla:

Generación	Adap. Media	Adap. Max.	x
1	0.257073	0.493223	0.8474
2	0.378307	0.498506	0.925525
3	0.395765	0.49323	0.847476
4	0.454126	0.499953	0.986408
5	0.458169	0.499953	0.986408
6	0.467791	0.499953	0.986408
7	0.483441	0.499953	0.986408
8	0.483225	0.499953	0.986408
9	0.489634	0.499997	1.00365
10	0.488957	0.499997	1.00357

Ejemplo 2: maximización de una función

- ❑ Podemos ver como a medida que avanzan las generaciones los valores medio y máximo de la adaptación de la población tienden a mejorar.
- ❑ Sin embargo, la mejora de la adaptación de una generación a otra no está garantizada, a menos que se introduzca elitismo
- ❑ El elitismo es una técnica para garantizar la supervivencia del mejor, o de algunos de los mejores, de generación en generación.
- ❑ Así, en nuestro ejemplo podemos ver que de la generación 2 a la 3 la adaptación del mejor individuo de la población baja de 0.498506 a 0.49323.

El algoritmo

```
algoritmo_Genético_main() {  
    TPoblacion pob;    // población  
    . . .  
    obtener_parametros(parametros);  
    pob = poblacion_inicial();  
    evaluación pob  
    // bucle de evolución  
    Para cada generación {  
        selección  
        cruce  
        mutación  
        evaluar pob  
    }  
    devolver pob[pos_mejor];  
}
```

```
Clase AGenetico {
```

```
    Cromosoma[] pob; // población de individuos
    entero tam_pob; // tamaño población
    entero num_max_gen; // número máximo de generaciones
    Cromosoma elMejor; // mejor individuo
    entero pos_mejor; // posición del mejor cromosoma
    real prob_cruce; // probabilidad de cruce
    real prob_mut; // probabilidad de mutación
    real precision; // precisión de la representación
                    // en algunos problemas
```

```
    . . .
```

El algoritmo

```
run() {  
    Iniciar poblacion  
    Evaluar población  
    Para cada generación {  
        //Selección  
        //Cruce  
        //Mutación  
        evaluar población  
    }  
    Devolver mejor  
}
```

```
... main(String[] args) {  
    . . .  
    AGenetico AG = new AGenetico();  
    AG.inicializa(); //crea población inicial de cromosomas  
    AG.evaluarPoblacion(); //evalúa los individuos y coge el mejor  
    while (!GA.terminado()) {  
        AG.numgeneracion++;  
        AG.seleccion();  
        AG.cruce();  
        AG.mutacion();  
        AG.evaluarPoblacion();  
    }  
    . . .  
    devolver pob[pos_mejor];  
}
```

El individuo: Cromosoma o individuo binario

Clase abstracta Cromosoma {

```
boolean[] genes; //cadena de bits (genotipo)
real fenotipo; //fenotipo
real aptitud; //función de evaluación fitness adaptación);
real puntuación; //puntuación relativa(aptitud/suma)
real punt_acum; //puntuación acumulada para selección
int longitud;
```

. . .
}

//normalmente codificamos más
de una variable o atributo, por
lo habrá que generalizar
Gen[] genes;
...

```
class Gen {
    boolean[] alelos;
```

0 1 1 0 1 0 0 1 1 1
X₁ X₂

```
public abstract class Individuo<T>
{
    T[] cromosoma;
    int[] tamGenes;
    . . .
    . . .
}
```

El individuo o Cromosoma

```
class CromosomaProblemaConcreto extends Cromosoma {  
    //Este puede ser un cromosoma o individuo concreto usado para  
    // maximizar o minimizar una función y utiliznado  
    // representación binaria de los genes  
    . . . .  
    longitudCromosoma = . . .  
    boolean maximizar;  
    . . .  
    double fenotipo() {. . .} //el valor que codifica el individuo  
  
    double evalua() { . . .} // adaptacion del individuo . . .  
                                // en este caso devuelve el fitness  
}
```


Generación de la población inicial

//crea los cromosomas y los inicializa aleatoriamente

inicializa() {

...

```
for (int j = 0; j < tam poblacion; j++) {  
    poblacion[j] = new CromosomaProblemaConcreto();  
    poblacion[j].inicializaCromosoma();  
    poblacion[j].aptitud=poblacion[j].evalua(); // aptitud
```

}

```
public void inicializaCromosoma() {  
    for (int i = 0; i < longitudCromosoma; i++) {  
        genes[i] = MyRandom.boolRandom();  
    }  
}
```

Si es < 0.5, asigna 0; caso contrario asigna 1

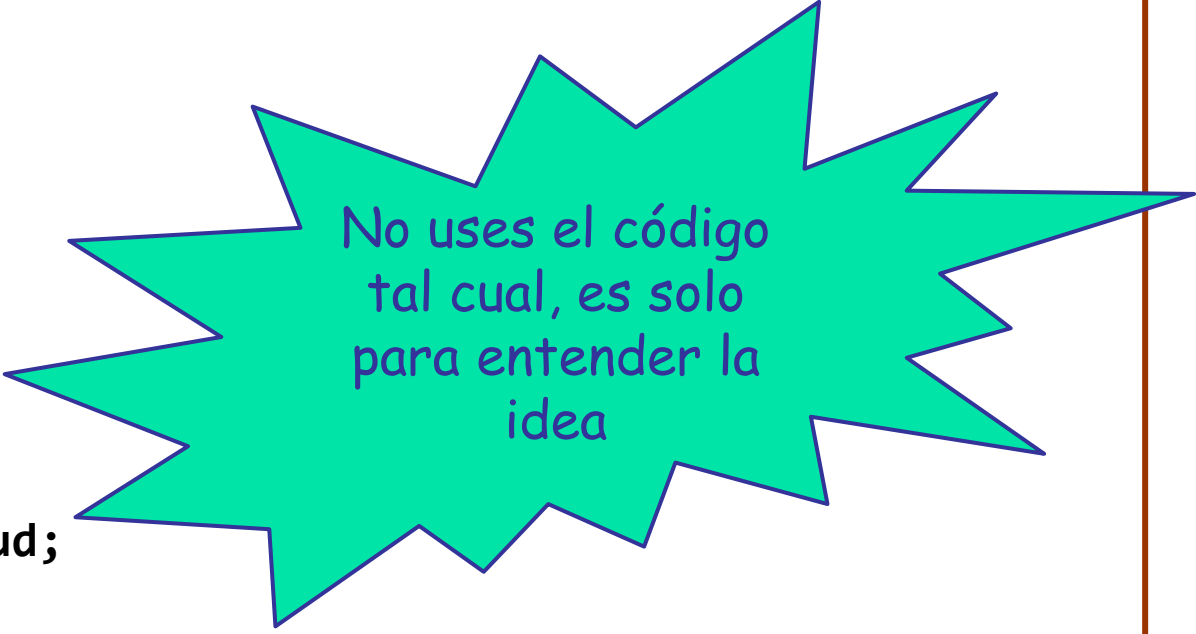
Función de adaptación (fitness)

```
Clase CromosomaProblemaConcreto {  
    . . .  
    double evalua( ){  
        //calcula fitness o adaptacion del cromosoma  
        double x; // fenotipo(o array de fenotipos sin son varias variables)  
        x = fenotipo();  
        devolver f(x); // f es la función a optimizar  
    }  
  
    double fenotipo( ){  
        //devuelve el valor que codifica el individuo  
    }  
}
```

Evaluación de la población

Se calcula el mejor individuo.

```
evaluarPoblacion() {  
    real punt_acu = 0; // puntuación acumulada  
    real aptitud_mejor = 0; // mejor aptitud  
    real sumaptitud = 0; // suma de la aptitud  
    . . .  
    para cada i desde 0 hasta tam_pob hacer {  
        sumaptitud = sumaptitud + poblacion[i].aptitud;  
        si (poblacion[i].aptitud > aptitud_mejor){  
            pos_mejor = i;  
            aptitud_mejor = poblacion[i].aptitud;  
        }  
    }  
}
```



No uses el código
tal cual, es solo
para entender la
idea

Aptitud == fitness

Evaluación de la población

Se revisa la aptitud relativa y puntuación acumulada de los individuos de la población.

```
para cada i desde 0 hasta tam_pob hacer {  
    pob[i].puntuacion = pob[i].aptitud / sumaptitud;  
    pob[i].punt_acu = pob[i].puntuacion + punt_acu;  
    punt_acu = punt_acu + pob[i].puntuacion;  
} }
```

```
// Si el mejor de esta generación es mejor  
// que el mejor que tenia de antes lo actualizo
```

```
if (aptitud_mejor > elMejor.dameAptitud()) {  
    elMejor ← pob[pos_mejor]  
}
```

Varía en caso de que sea una minimización.

Individuo	Fitness u_i	puntuación p_i	acumulada q_i
1	4	0,2	0,2
2	1	0,05	0,25
3	1	0,05	0,3
4	2	0,1	0,4
5	3	0,15	0,55
6	2	0,1	0,65
7	5	0,25	0,9
8	2	0,1	1
	20		

- ❑ La función de selección escoge un número de supervivientes **igual al tamaño** de la población.
- ❑ La función modifica la población que pasa a estar formada únicamente por ejemplares de los individuos supervivientes.
- ❑ Antes de hacer la selección hemos calculado las puntuaciones de cada individuo:
 - **aptitud** (fitness)
 - **puntuación** ($\text{fitness} / \text{suma_fitness}$)
 - **puntuación acumulada**

Presión Selectiva

Es la “**Fuerza**” del mecanismo de la selección:

$$\frac{fitnessMaximo}{fitnessMedio}$$

$$5/2.5 = 2$$

Individuo	Fitness u_i	puntuación p_i	acumulada q_i
1	4	0,2	0,2
2	1	0,05	0,25
3	1	0,05	0,3
4	2	0,1	0,4
5	3	0,15	0,55
6	2	0,1	0,65
7	5	0,25	0,9
8	2	0,1	1
	20		

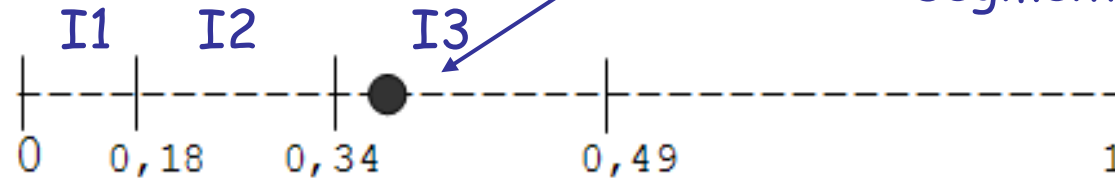
*TamañoPoblación * Puntuación Máxima*

$$8 * 0.25 = 2$$

- Si la presión selectiva es baja los valores de aptitud están muy cerca entre sí y se produce un estancamiento de la búsqueda.
- La **Convergencia prematura** se produce cuando los individuos muy aptos dominan la población rápidamente. Óptimos locales.

Selección por ruleta

```
void selecciónRuleta ( pob, nuevaPob, tam_pob) {  
entero sel_super[tam_pob]; //seleccionados para sobrevivir  
real prob; // probabilidad de seleccion  
entero pos_super; // posición del superviviente  
para cada i desde 0 hasta tam_pob hacer {  
    prob = alea(); → 0,38  
    pos_super = 0;  
    mientras ((prob > pob[pos_super].punt_acu) y  
              (pos_super < tam_pob)) pos_super++;  
    sel_super[i] = pos_super;  
}
```

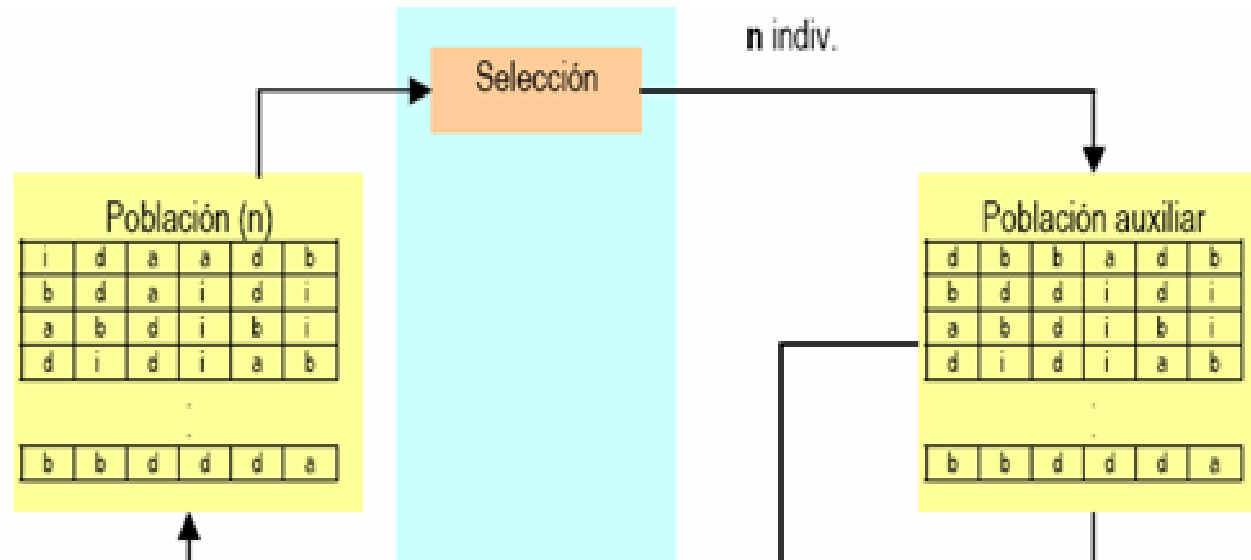
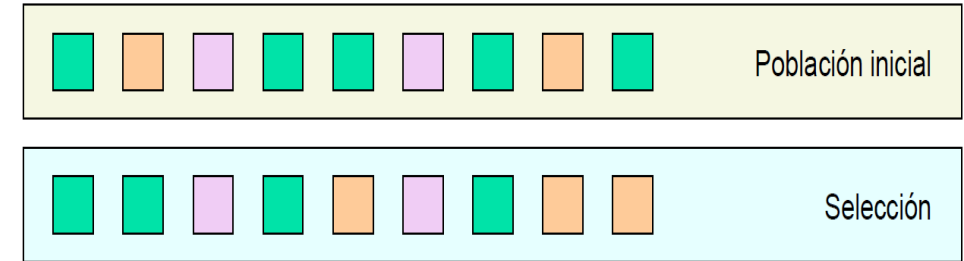


Se busca el lugar que le corresponde a 0.38 y se selecciona el individuo correspondiente a ese segmento: I3

Selección

// se genera la poblacion intermedia

```
para cada i desde 0 hasta tam_pob hacer {  
    copiar (pob[sel_super[i]], nuevaPob);  
}
```

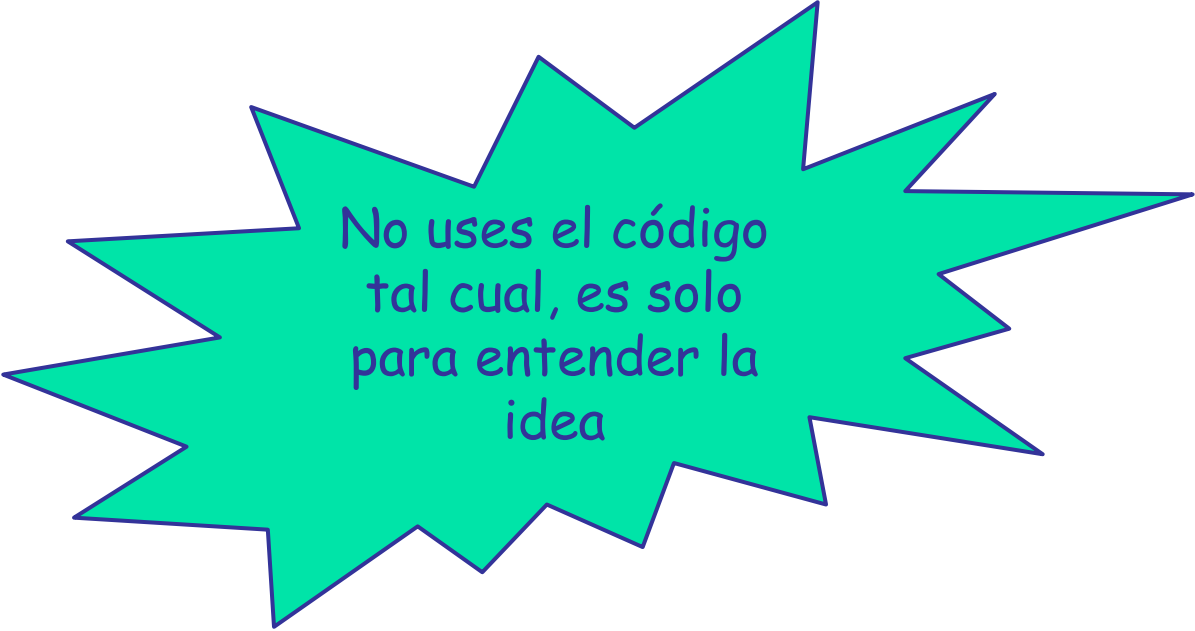


Selección: diferentes métodos

```
public class FactoriaSeleccion {  
  
    static AlgoritmoSeleccion getAlgoritmoDeSeleccion(String algoritmo, int participantes)  
    {  
        switch(algoritmo) {  
            case "Ruleta" :      return new Seleccion_Ruleta();  
            case "Torneo_D":     return new Seleccion_Torneo(participantes);  
            case "Torneo_P":     return new Seleccion_TorneoProb(participantes);  
            case "Ranking" :     return new Seleccion_Ranking();  
  
            default:  
                return new Seleccion_Ruleta();  
        }  
    }  
}
```

Reproducción: cruce

```
cruce( ) {  
    // cruce monopunto  
  
    //seleccionados para reproducir  
    entero sel_cruce[tam_pob];  
  
    //contador seleccionados  
    entero num_sele_cruce = 0;;  
    entero punto_cruce;  
    real prob;  
    Cromosoma hijo1,hijo2;
```



No uses el código
tal cual, es solo
para entender la
idea

Reproducción: cruce

```
//Se eligen los individuos a cruzar
para cada i desde 0 hasta tam_pob {
    //se generan tam_pob números aleatorios en [0 1)
    prob = alea();
    //se eligen los individuos de las posiciones i si prob < prob_cruce
    si (prob < prob_cruce){
        sel_cruce[num_sel_cruce] = i;
        num_sel_cruce++;
    }
}
// el numero de seleccionados se hace par
si ((num_sel_cruce mod 2) == 1)
    num_sel_cruce--;
```

Reproducción: cruce

```
// se cruzan los individuos elegidos en un punto al azar
```

```
punto_cruce = alea_ent(0,lcrom);
```

```
para cada i desde 0 hasta num_sel_cruce avanzando 2{
```

```
    cruce(pob[sel_cruce[i]], pob[sel_cruce[i+1]], hijo1, hijo2, punto_cruce, );
```

```
// los nuevos individuos sustituyen a sus progenitores
```

```
pob[sel_cruce[i]] = hijo1;
```

```
pob[sel_cruce[i+1]] = hijo2;
```

```
}
```

```
}
```

Reproducción: cruce

```
cruce(padre1,padre2,hijo1,hijo2, puntoCruce){  
    entero i;  
    hijo1.genes.iniciar();  
    hijo2.genes.iniciar();  
    // primera parte del intercambio: 1 a 1 y 2 a 2  
    para cada i desde 0 hasta punto_cruce hacer{  
        hijo1.genes.insertar(padre1.genes[i]);  
        hijo2.genes.insertar(padre2.genes[i]);  
    }  
}
```

Reproducción: cruce

// segunda parte: 1 a 2 y 2 a 1

```
para cada i desde punto_cruce hasta lcrom; hacer{  
    hijo1.genes.insertar(padre2.genes[i]);  
    hijo2.genes.insertar(padre1.genes[i]);  
}
```

// se evalúan

```
hijo1.aptitud = evalua(hijo1, . . . );  
hijo2.aptitud = evalua(hijo2, . . . );  
}
```

Código mejorado...

```
public void crossOver(genotype other){  
    // Crosses this and other genotype  
    int point_x = ((int)(Math.random()*(data_x.length)));  
    int point_y = ((int)(Math.random()*(data_y.length)));  
    for (int i=point_x;i<data_x.length;i++){  
        boolean tmp = this.data_x[i];  
        this.data_x[i]=other.data_x[i];  
        other.data_x[i]=tmp;  
    }  
    for (int i=point_y;i<data_y.length;i++){  
        boolean tmp = this.data_y[i];  
        this.data_y[i]=other.data_y[i];  
        other.data_y[i]=tmp;  
    }  
}
```

- ❑ El operador de mutación considera la posible mutación de cada gen del genotipo con la probabilidad indicada.
- ❑ La función revisa la aptitud del individuo en caso de que se produzca alguna mutación.

mutacion(pob, tam pob, prob mut, . . .){

booleano mutado;

entero i,j;

real prob;


```
para cada i desde 0 hasta tam_pob hacer{
    mutado = false;
    para cada j desde 0 hasta lcrom hacer{
        // se genera un numero aleatorio en [0 1)
        prob = alea();
        // mutan los genes con prob<prob_mut
        si (prob<prob_mut){
            pob[i].genes[j] = not( pob[i].genes[j]);
            mutado = true;
        }
        si (mutado)
            pob[i].aptitud = pob[i].evalua();
    }
}
```

Ejemplo: maximización de una función

Maximizar

$$f(x) = 21.5 + x \cdot \text{sen}(4\pi x)$$

que presenta un máximo de en $x =$

$$x \in [-3.0, 12.1]$$

0	1	1	0	1	0	0	1	1	1
---	---	---	---	---	---	---	---	---	---

x

- **Representación de los individuos:** se representan mediante cadenas binarias que se corresponden con los puntos del espacio de búsqueda

Cromosoma concreto: maximización de una función

```
class CromosomaProblemaConcreto extends Cromosoma {  
    . . . .  
    // extremos del intervalo considerado  
    // para los valores del dominio  
    xmin= . . .;  
    xmax=. . .;  
  
    . . . .  
    longitudCromosoma = . . .  
    boolean maximize = true;  
    . . .  
    double fenotipo() { . . . } //el valor del individuo  
    double evalua() { . . . }  
    // adaptacion del individuo..en este caso devuelve  
    // el fitness de dicho individuo  
    . . . .  
}
```

Cálculo de longitud de cromosoma

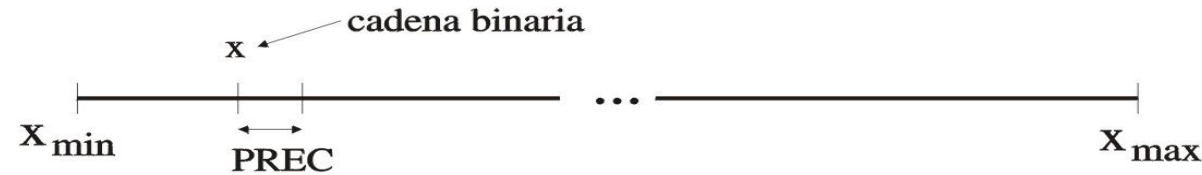
- ❑ Buscamos una codificación: representación del dominio del problema mediante enteros binarios sin signo.
- ❑ La elección más sencilla consiste en discretizar el intervalo $[x_{min}, x_{max}]$ en una cantidad de 2^{lcrom} puntos tal que la distancia entre puntos consecutivos sea menor que la tolerancia especificada,

$$\frac{x_{max} - x_{min}}{2^{lcrom} - 1} < TOL$$

- ❑ Cada punto del espacio de búsqueda queda representado mediante un entero binario de longitud $lcrom$, comenzando por 0...0 que representa a x_{min} y terminando en 1...1 que representa a x_{max}

Cálculo de longitud de cromosoma

- Buscamos una codificación: representación del dominio del problema mediante enteros binarios sin signo.



- La elección más sencilla consiste en discretizar el intervalo $[x_{\min}, x_{\max}]$ en una cantidad de 2^{lcrom} puntos tal que la distancia entre puntos consecutivos sea menor que la tolerancia especificada,

$$\frac{x_{\max} - x_{\min}}{2^{lcrom} - 1} < TOL$$

- Cada punto del espacio de búsqueda queda representado mediante un entero binario de longitud $lcrom$, comenzando por 0...0 que representa a x_{\min} y terminando en 1...1 que representa a x_{\max}

Cálculo de longitud de cromosoma

- La longitud de los individuos se calcula a partir de la tolerancia TOL :

$$lcrom = \left\lceil \log_2 \left(1 + \frac{x_{max} - x_{min}}{TOL} \right) \right\rceil$$

- Recíprocamente se puede calcular el punto x que corresponde a un individuo v mediante la siguiente fórmula de decodificación:

$$x(v) = x_{min} + \text{bin2dec}(v) \cdot \frac{x_{max} - x_{min}}{2^{lcrom} - 1}$$

Ejemplo

Buscar el punto x_1, x_2 que maximiza la función:

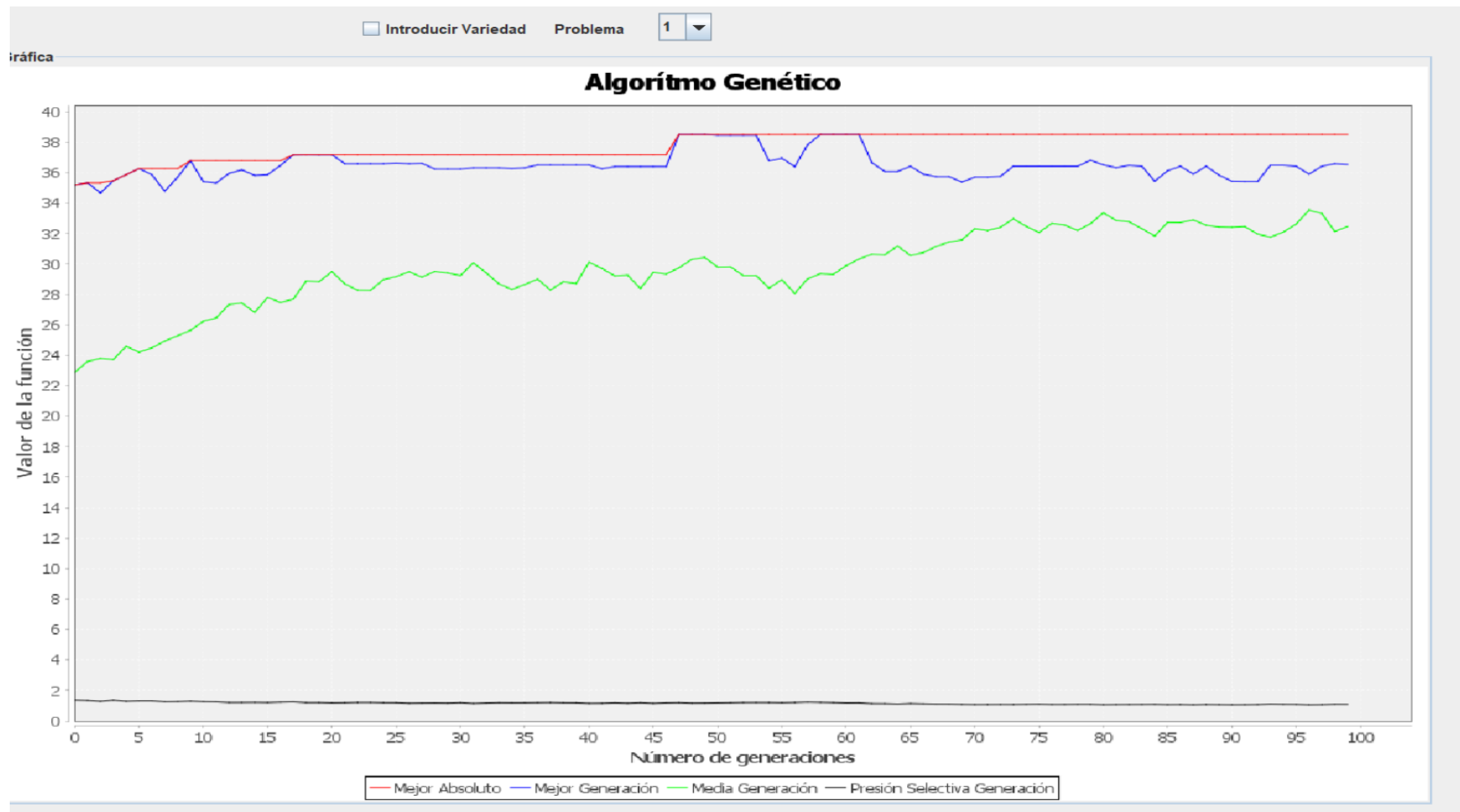
$$f(x_1, x_2) = 21.5 + x_1 \cdot \text{sen}(4\pi x_1) + x_2 \cdot \text{sen}(20\pi x_2)$$

$$x_1 \in [-3.0, 12.1] \quad x_2 \in [4.1, 5.8]$$

0	1	1	0	1	0	0	1	1	1
x_1							x_2		

Dicha función presenta un máximo de **38.809** en 11.625 y 5.726

Gráfica de evolución



Ejemplos de código para un algoritmo genético

```
public class AllOnesGA {
    public static void main(String[] args) {
        // Create GA object
        GeneticAlgorithm ga = new GeneticAlgorithm(100, 0.001, 0.95, 2);
        Population population = ga.initPopulation(50); // Initialize population
        ga.evalPopulation(population); // Evaluate population
        // Keep track of current generation
        int generation = 1;
        while (ga.isTerminationConditionMet(population) == false) {
            // Print fittest individual from population
            System.out.println("Best solution: " + population.getFittest(0).toString());
            population = ga.crossoverPopulation(population); // Apply crossover
            population = ga.mutatePopulation(population); // Apply mutation
            ga.evalPopulation(population); // Evaluate population
            generation++; // Increment the current generation
        }

        System.out.println("Found solution in " + generation + " generations");
        System.out.println("Best solution: " + population.getFittest(0).toString());
    }
}
```

```
public class GeneticAlgorithm {
    private int populationSize;
    private double mutationRate;
    private double crossoverRate;
    private int elitismCount;

    . . .
}
```

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

APress

Ejemplos de código para un individuo más genérico

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

APress

```
public abstract class Chromosome {  
    protected List<Gene> genes;  
    protected List<Double> fenotype;  
  
    protected double fitness; //funcion de evaluacion fitness  
    protected double score; //puntuacion relativa (aptitud/suma)  
    protected double scoreAccumulated; //puntuacion acumulada para seleccion  
    private String tipo; //tipo de cromosoma  
  
    //extremos del intervalo para los valores del dominio  
    //protected double xmin, xmax;  
    protected int chromosomeLength; //longitud del cromosoma  
    protected double tolerance;  
    protected double adaptation;  
    protected double escalation;
```

Ejemplos de código para un individuo más genérico

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

Apress

```
public Chromosome(double min, double max, double tolerance) {  
    this.xmin = min;  
    this.xmax = max;  
    this.tolerance = tolerance;  
}  
  
public List<? extends Gene> getGenes(){  
    return this.genes;  
}  
  
public Gene getGene(int pos){  
    return this.genes.get(pos);  
}
```

Ejemplos de código para un individuo más genérico

```
public abstract class Gene {  
    protected List<Object> allele;  
    protected int geneLength;  
  
    public Gene(int geneLength) {  
        this.allele = new LinkedList();  
        this.geneLength = geneLength;  
    }  
  
    public int getLength(){  
        return geneLength;  
    }  
  
    public Object getAllele(int pos){  
        return allele.get(pos);  
    }  
}
```

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

APress

Ejemplos de código para un individuo más genérico

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

APress

```
public class BinaryGene extends Gene{

    public BinaryGene(int geneLength) {
        super(geneLength);
        this.allele = new LinkedList();
    }

    @Override
    public void initializeGene(Random randomNumber) {
        for(int i=0; i < this.getLength(); i++){
            this.allele.add(randomNumber.nextBoolean());
        }
    }
}
```

Ejemplos de código para un individuo más genérico

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

Apress

```
public class RealGene extends Gene{  
    private double min;  
    private double max;  
  
    public RealGene(int geneLength, double min, double max) {  
        super(geneLength);  
        this.min=min;  
        this.max=max;  
    }  
  
    @Override  
    public void initializeGene(Random randomNumber) {  
        this.allele.add(ThreadLocalRandom.current().nextDouble(this.min, this.max));  
    }  
}
```

```
public AGView executeAlgorithm() {  
    this.initialize();  
    this.evaluate();  
    while (currentGeneration != maxGenerations) {  
        if (elitism) {  
            this.eliteChromosomes.addAll(0, this.elite.getElite(population));  
        }  
  
        this.selection();  
        this.crossover();  
        this.mutate();  
        . . .  
    }  
}
```

Genetic Algorithms in Java Basics

Lee Jacobson, Burak Kanber

Apress

```
this.mutate();  
  
. . .  
if (elitism) {  
    this.population = this.elite.includeEliteRepWorst(this.population, this.eliteChromosomes);  
    this.eliteChromosomes.clear();  
}  
this.evaluate();  
this.generationAverage[this.currentGeneration] = getGenerationAvg();  
this.generationBest[this.currentGeneration] = getGenerationBest();  
this.absoluteBest[this.currentGeneration] = getAbsoluteBest();  
currentGeneration++;  
}  
AGView viewInfo = new AGView(this.generationAverage, this.generationBest, this.absoluteBest);  
return viewInfo;  
}
```