Práctica 1 – Estrategias Evolutivas y GAs

Asignatura: Soft Computing

Titulación: Ingeniería en Sistemas de Información

Autores: Iván Sánchez Ranz - Walter Huatay Rodríguez

Ejercicio 1

<u>Estrategia Evolutiva (1+1)-EE:</u>

Esta estrategia evolutiva es la más básica, ya que parte de un único individuo en la población, el cual genera una descendencia (1 hijo) mutando, luego se selecciona al mejor de ambos individuos pudiéndose dar que el padre sobreviva o que el hijo sustituya al padre.

A continuación se muestra la captura del código junto con su correspondiente explicación:

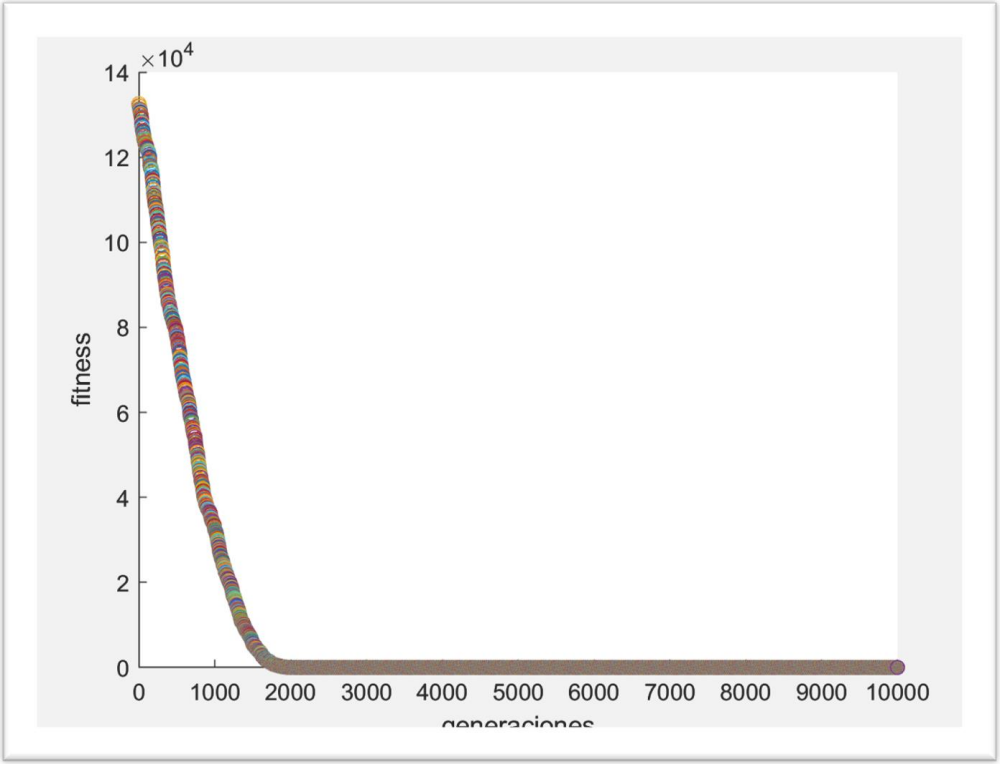
```
5
6   sigma=0.5; % Establecemos la variable sigma para la mutación en 0,5
7   padre = -100 + (100+100)*rand(1,30); % Se crea al padre con valores entre -100 y 100.
8   fitnessPadre = sum(padre.^2); %Almacenamos el valor fitness del padre.
9   iteracion = 10000; %Creamos las iteraciones (generaciones) durante las que tendrá lugar esta estrategia evolutiva
10
11  %Preparamos la representación gráfica de la estrategia evolutiva
12  figure;
13  xlabel("generaciones"); % Eje Y
14  ylabel("fitness");  % Eje X
15  hold on;
16
17  for i= 1:iteracion % Creamos el bucle que se realizará durante todas las generaciones.
18     hijo = padre + sigma*randn(1,30); % Creamos el hijo a partir del padre gracias al sigma.
19     fitnessHijo = sum(hijo.^2);   %Almacenamos el valor fitness del hijo
20     if (fitnessHijo < fitnessPadre) % Establemos la condición: Si el fitness del hijo es menor que el del padre.
21      fitnessPadre = fitnessHijo; %Si el fitness del hijo es menor (mejor) se almacena como fitness padre.
22      padre=hijo; % Y el padre pasaría a ser el hijo superviviente al tener mejor fitness.
23     end
24     plot(i,fitnessPadre,"-o"); % Se representa la variable fitnessPadre, la cual es la mejor de cada iteración.
25  end
26  hold off;
```

- En primer lugar, definimos el sigma que será utilizado para la mutación del padre para generar a los hijos. Tras esto se crea al padre con valores entre -100 y 100. Una vez creado el padre, almacenamos su valor fitness en una variable y creamos la variable que contiene el número de iteraciones (generaciones) que tendrán lugar en esta estrategia evolutiva.

- En segundo lugar preparamos los componentes de la interfaz gráfica, tanto su Eje X que representa el número de generaciones (iteraciones), como el Eje Y que representa el valor fitness a lo largo de las generaciones.

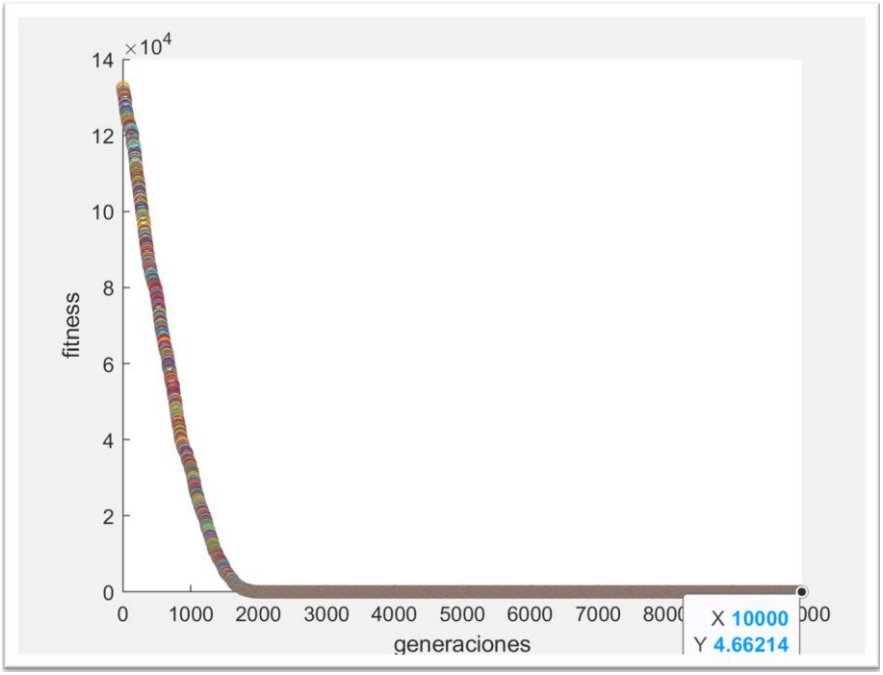
- Por último, y lo más importante, creamos el bucle el cual repetirá durante 10.000 generaciones lo siguiente: Si el valor fitness del hijo es mejor que el del padre será el hijo el que sobrevivirá y su valor fitness pasará a ser el valor fitness padre que será comparado con el siguiente hijo, y así durante todo el bucle, de manera que si el hijo es mejor irá sustituyendo al padre generación tras generación.

Respecto a la **representación gráfica** que muestra el **resultado final**:

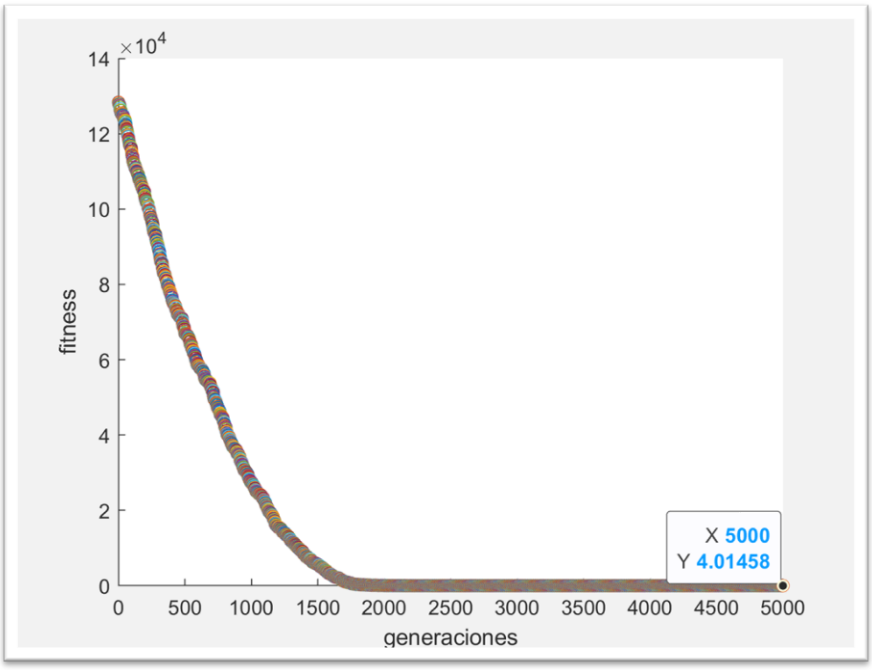


En la representación gráfica se observa que el valor fitness se va reduciendo de manera radical hasta la generación 2000, lo cual quiere decir que generación tras generación el mejor fitness se va imponiendo de manera contundente hasta que a partir de la generación 2000 las nuevas generaciones ya no son tan buenas y se estabiliza a lo largo del resto de generaciones.

Además, respecto al valor fitness al final de las 10.000 generaciones tendremos un valor fitnes de **4,66214.** Como se muestra en la siguiente imagen:



Recordemos que el objetivo es reducir el valor fitness a un valor lo más cercano posible a cero, por tanto, ahora se prueba con 5000 generaciones, aquí obtenemos un fitness final de 4,01458, lo cual es un mejor fitness que en el caso anterior. Se muestra en la siguiente imagen:



Estrategia evolutiva (υ + λ)-EE:

Mediante esta estrategia evolutiva se crean **λ** hijos a partir de **υ** padres, se van eliminando de manera determinista los peores.

A continuación se muestra el código realizado:

```
lambda = 20; %Numero de hijos
mu = 5; % Número de padres
sigma = 0.3; %Establecemos la variable sigma
iteraciones = 5000; %Establecemos el número de iteraciones (generaciones)
padre = -100 + (100+100)*rand(1,30); %Creamos el padre.
vector_fitness = zeros(1,iteraciones);  %Vector para almacenar fitness

figure;
xlabel("generaciones");
ylabel("fitness");
hold on;

for i = 1:iteraciones
    for index_padre = 1:size(padre,1)
       hijos = padre(index_padre,:) + sigma.*randn(mu,size(padre,2));
       fitness_hijos = sum(hijos.^2,2);
       [minimum,mejor_hijo] = min(fitness_hijos);

       if minimum < sum(padre(index_padre,:).^2)
           padre(index_padre,:) = hijos(mejor_hijo,:);
       end
    end

    fitness_padres = sum(padre.^2,2);
    [menor_fitness,mejor_solucion] = min(fitness_padres);
    vector_fitness(i) = menor_fitness;
    plot(i, menor_fitness,"-o");
end
```

En primer lugar, creamos las variables que se utilizarán en el algoritmo: a destacar **mu** y **lambda**, la primera de ellas es el número de hijos y la segunda el número de padres. Además, se crea el padre, se establece el número de iteraciones a realizar y el vector donde se irán guardando los mejores fitness.

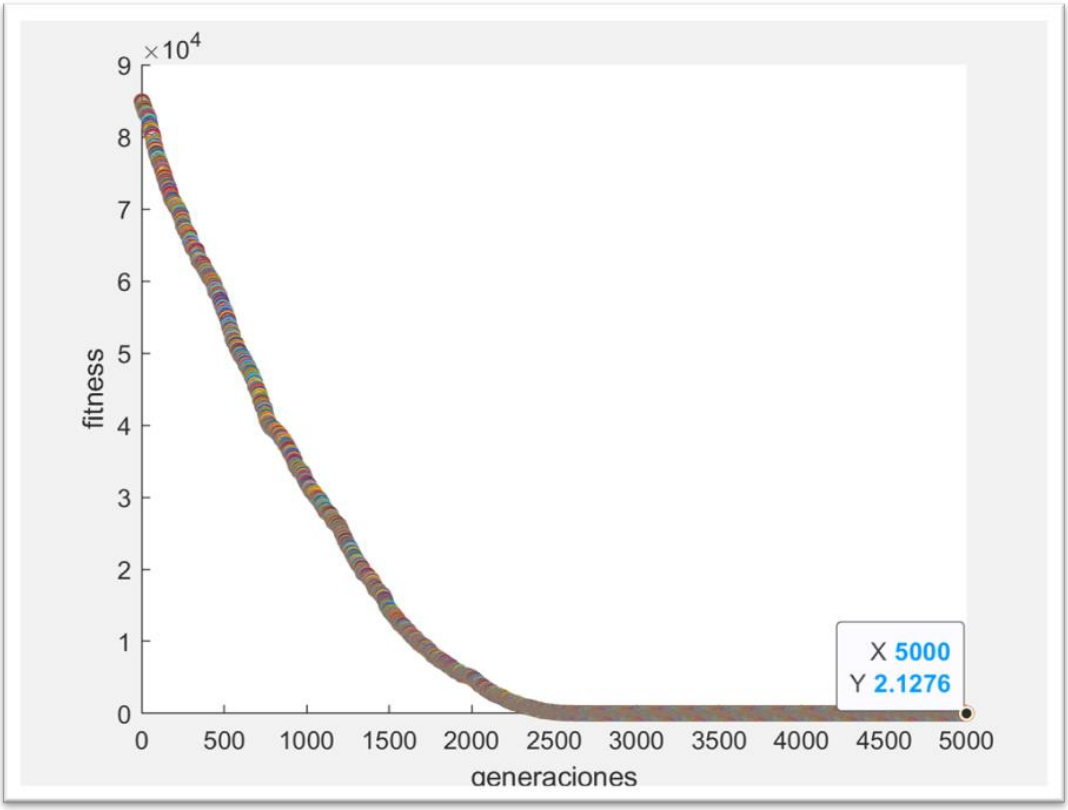
Tras esto establecemos nuevamente lo necesario para la representación gráfica, como en el caso anterior en el Eje X están las generaciones y en el Eje Y están los valores fitness.

Posteriormente, está el bucle anidado, en el cual se recorre el vector del padre y se crearán los hijos a partir de los padres y gracias al sigma, y almacenamos el valor del fitness de los hijos en una variable. Además, en este bucle se actualiza el vector de padres, generando, en este caso, 5 hijos por cada padre.

Se irá sustituyendo al padre si el mejor de los hijos tiene un mejor fitness. Para decidir quién es el mejor hijo nos quedamos con aquel que tenga el mínimo fitness, es decir, el valor más pequeño de fitness. Así que si el mínimo de los fitness es mejor que el padre entonces se intercambia el padre por ese hijo.

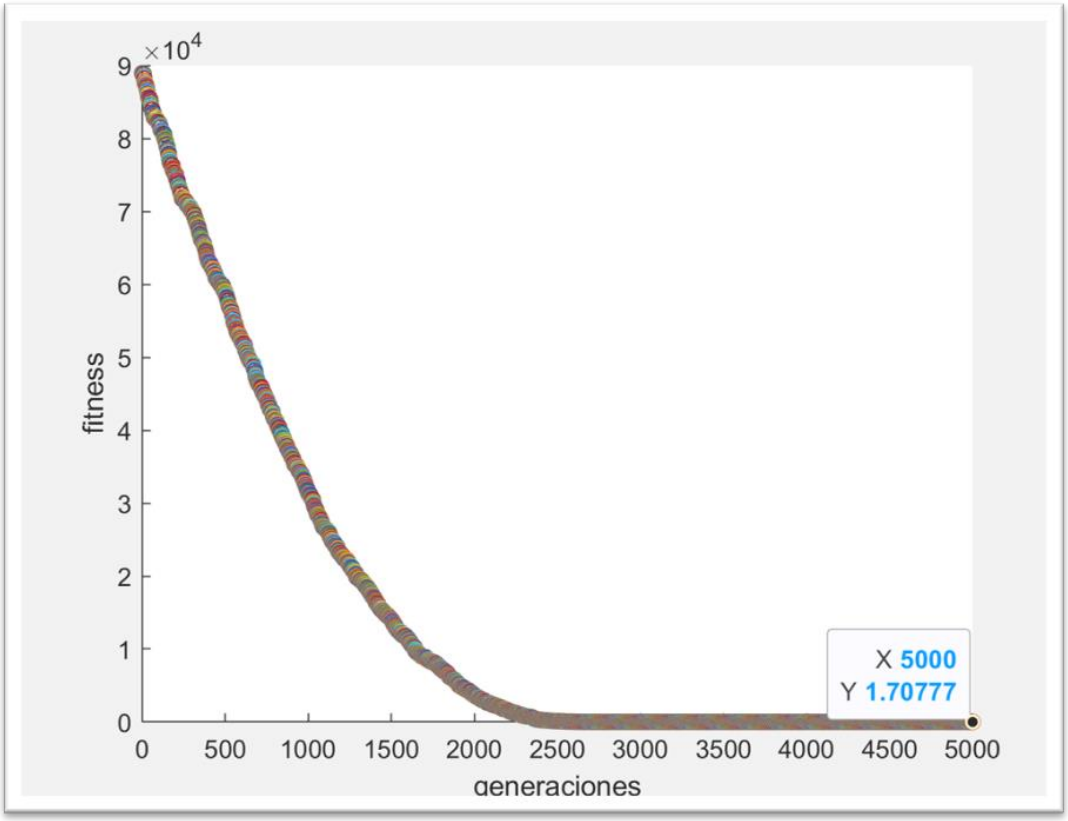
Por último, se ha creado la matriz fitness_padres, la cual almacena los fitness de cada iteración. De esta matriz seleccionaremos el menor fitness el cual se almacenará en el vector_fitness, por tanto, este vector al final del algoritmo habrá almacenado los menores fitness. Finalmente mostraremos en la gráfica los mejores fitness que se han encontrado a lo largo del algoritmo.

A continuación, se muestra la gráfica del resultado del algoritmo:



Se puede observar que se produce una gran reducción del valor de los fitness hasta, aproximadamente, la generación 2500, a partir de la cual se equilibra hasta que finalmente se obtiene un valor fitness de 2,1276 en la generación 5000.

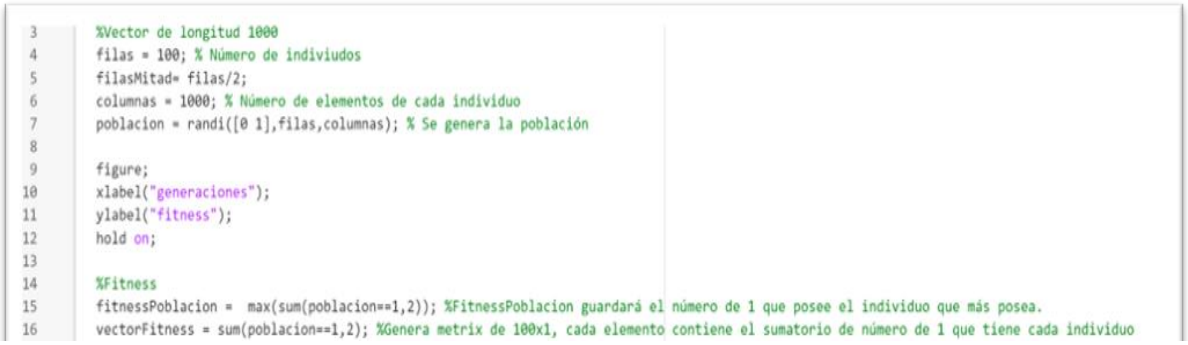
Se ha decidido volver a ejecutar el algoritmo con el mismo número de iteraciones (5000) pero en este caso se han establecido que sean 10 padres y cada uno de ellos tenga 30 hijos, de esta forma se consigue reducir el fitness a 1,7077. Se muestra a continuación:



Ejercicio 2

Algoritmo Genético.

El algoritmo genético realizado está formado por una selección, un cruce y una mutación a lo largo de las generaciones.
A continuación, se muestra el código implementado y la explicación de todos los pasos realizados a lo largo de éste:



```
%Vector de longitud 1000
filas = 100; % Número de individuos
filasMitad= filas/2;
columnas = 1000; % Número de elementos de cada individuo
poblacion = randi([0 1],filas,columnas); % Se genera la población

figure;
xlabel("generaciones");
ylabel("fitness");
hold on;

%Fitness
fitnessPoblacion =  max(sum(poblacion==1,2)); %FitnessPoblacion guardará el número de 1 que posee el individuo que más posea.
vectorFitness = sum(poblacion==1,2); %Genera metrix de 100x1, cada elemento contiene el sumatorio de número de 1 que tiene cada individuo
```

En esta primera imagen, establecemos los datos que se mencionan en el enunciado. Partimos de una matriz de 100 individuos, cada una contiene 1000 elementos, a partir de esto se creará la población inicial. El objetivo es llegar a tener el máximo número de unos en la matriz.

Además, establecemos el eje X (generaciones) y el eje Y (fitness) que forman parte de la representación gráfica donde se observará la evolución de la población a lo largo de las generaciones.

A continuación, se crea el vector fitnessPoblación donde se guarda la cantidad de números 1 que posee el individuo que más posea. El vectorFitness se trata de una matriz de 100x1 en la que cada elemento contiene el sumatorio de número 1 que posee cada individuo.

```
for i=1:5000
    vectorFitness = sum(poblacion==1,2);
    fitnessPoblacion1 =  max(sum(poblacion==1,2));
    total1 = sum(sum(poblacion==1,2));

    %Seleccion
    longitud = length(vectorFitness);
    [lista,indices] = sort(vectorFitness,'descend'); %Ordenamos por orden descendente a los valores fitness
    indicesSeleccionados = indices(1:longitud/2); % Se seleccionan los mejores 50 individuos basados en el fitness
    selec = poblacion(indicesSeleccionados,:); %Almacenamos los 50 mejores individuos que serán utilizados para obtener los individuos de la poblacion

    poblacionSeleccionada = selec;
    poblacion(1:filasMitad,1:columnas) = poblacionSeleccionada;


    %Cruce - generamos las poblaciones resultantes de los cruces entre
    %individuos:
    poblacionCruzada = cruzar(poblacion,poblacionSeleccionada);
    poblacion(filasMitad+1:filas,1:columnas) = poblacionCruzada;
```

En la segunda imagen se inicia el bucle, en este caso se tendrán 5000 generaciones.

Al principio de cada generación se realizan los mismos cálculos de fitness que hemos visto anteriormente.

Una vez dentro del bucle será el momento de realizar la selección, cruzamiento y mutación. En primer lugar, está la "Selección", para ello ordenaremos los individuos en orden descendente a partir de su valor fitness, mediante el comando sort. Una vez ordenados, se seleccionan los 50 primeros individuos, los cuales son los 50 mejores individuos de la población ya que están ordenados.
Estos 50 individuos sustituirán a los primeros 50 individuos de la población de la que hemos partido.

Tras esto, es el turno de realizar el "Cruce", para ello se ha creado una función aparte llamada "cruzar".

Función Cruzar:

```
function cruzar = cruzar(m,poblacionSeleccionada)
    long = length(m(1,:));
    filas = length(poblacionSeleccionada(:,1));
    cruzar = zeros(filas,long);
    descendencia = zeros(2,long);
    for i=1:2:length(poblacionSeleccionada(:,2))
        %Se seleccionan los dos primeros padres
        padre1 = poblacionSeleccionada(i,:);
        padre2 = poblacionSeleccionada(i+1,:);

        %En el vector descendencia se encuentran los hijos de los padres, que se
        %han cruzado

        %Hijo 1
        descendencia(1,1:long/2) = padre1(1:long/2);
        descendencia(1,long/2 + 1:long) =padre2(long/2 + 1:long);

        %Hijo 2
        descendencia(2,1:long/2) = padre2(1:long/2);
        descendencia(2,long/2 + 1:long) =padre1(long/2 + 1:long);

        %Se añade al vector cruzados, que es el que devolveremos
        cruzar(i,1:long) = descendencia(1,:);
        cruzar(i+1,1:long) = descendencia(2,:);
    end
end
```

En esta función se guarda en variables la longitud de la población inicial, el número de filas que define el número de individuos a crear, y la matriz descendencia que almacenará los hijos resultantes del cruce.

Ahora se recorre en bucle los individuos que forman la población de la que seleccionaremos 2 padres. El primero de los hijos se crea a partir de la unión de la primera mitad de un padre y de la segunda parte del otro padre. Mientras que el segundo hijo se crea a partir de la unión de los elementos que no hemos utilizado en la creación del primer hijo.

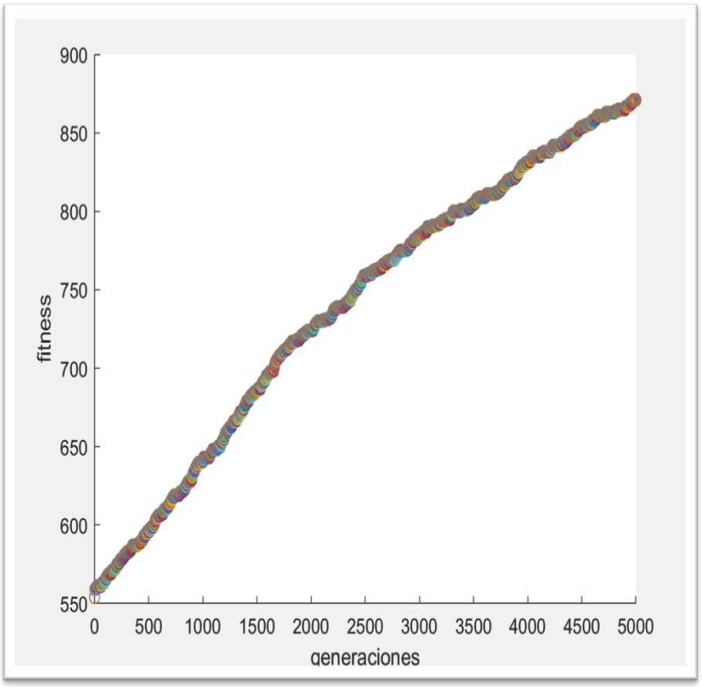
Finalmente, esta función devolverá una matriz que contenga los hijos que han sido generados.

```
39        %Mutacion
40        umbralMutacion=0.3;  %Definimos umbral de mutación.
41        probabilidadMutacion = rand(); %Definimos la probabilidad aleatoria entre 0 y 1.
42        if(umbralMutacion>probabilidadMutacion) %Por debajo del umbral se producirá mutación.
43              aleatorio_PosicionColumnas = randi([1 columnas]); % Se elige el elemento (posicion) a mutar.
44              aleatorio_PosicionFilas = randi([1 filas]);
45              poblacion(aleatorio_PosicionFilas,aleatorio_PosicionColumnas) = ...
46                1 - poblacion(aleatorio_PosicionFilas,aleatorio_PosicionColumnas);
47
48        end
49        fitnessPoblacion =  max(sum(poblacion==1,2));
50
51        plot (i,fitnessPoblacion, "-o");
52    end
```

En esta última imagen del código principal tiene lugar la "Mutación", en primer lugar, se define la variable umbral de mutación que se establece en 0.3, y la probabilidad de mutación que será un valor aleatorio entre 0 y 1.

Posteriormente, sólo se producirá la mutación si la probabilidad de mutación es inferior al umbral de mutación. Si esto se cumple se seleccionará de manera aleatoria el elemento que mutará.

Por último, se obtiene el vector fitnessPoblación que contiene el máximo fitness de la población a lo largo de cada generación, lo cual es lo que saldrá representado en la gráfica.



Finalmente obtenemos la gráfica final, esta representa como a lo largo de todas las generaciones el valor fitness va en aumento gracias a todas las operaciones realizadas.

Podemos concluir que, gracias al algoritmo, las generaciones generadas son continuamente mejores que la anterior, haciendo que el valor fitness aumente con cada generación.