Sistemas Inteligentes 1

Práctica 1
Algoritmos Genéticos. SGA

Héctor Garbisu Arocha
Curso 2015-2016
Grado en Ingeniería Informática
Universidad de las Palmas de Gran Canaria

Índice

1. Presentación del problema

Tenemos que diseñar un algoritmo para detectar elipses en una imagen.
Para que sea más fácil, la imagen será en blanco y negro y de 100x100 píxeles en todos los casos.
Utilizaremos el esquema general del algoritmo genético, con algunos de sus parámetros previamente fijados, como el método de cruce y la presencia de elitismo.

2. Esquema del SGA

Para tener una versión sin optimizar del algoritmo tenemos que garantizar que se cumplen las siguientes funciones en este orden:

Generar población, cálculo del fitness de cada individuo, selección, cruce, mutación y vuelta al segundo paso.

Cada paso tiene sus propias características. Algunas de ellas serán modificadas más adelante para optimizar el algoritmo, mientras que otras no.

2.1. Características generales del algoritmo genético

La generación aleatoria de cromosomas tendrá una distribución uniforme.
Cada gen tendrá información útil en cada bit (el valor máximo viable será todos los bits a uno).
El cruce será two point crossover, con una ventana intermedia de al menos un bit (se garantiza que cualquier cruce produce mezclas).
Habrá una mutación como máximo por cada cromosoma.

Codificación del cromosoma:
Los cromosomas tendrán cinco genes: Dos genes de 5 bits, dos genes de 6 bits y un gen de 7 bits.
Los dos primeros genes representarán los radios de la elipse, los dos genes de 6 bits codifican la posición (el centro) de la elipse dentro de la imagen y el último gen, codificará el ángulo de la elipse. En total son 29 bits por cromosoma.

2.2. Características ajustables del SGA

Los parámetros del algoritmo que se podrán modificar, y que permitirán optimizarlo, son los siguientes:

- Tamaño de la población
- Condición de parada (número de iteraciones)
- Probabilidad de mutación
- Cantidad de elitismo y poda
- Función de fitness

3. Implementación

3.1 Cuerpo principal - SGA1.m

El cuerpo principal del algoritmo está escrito en un script de Matlab. En un futuro podría ser una función a la que se le pasan como parámetros los hiperparámetros del algoritmo, pero de momento está bien así.
Al principio están las asignaciones de los valores del algoritmo, estos son los números que tenemos que optimizar más adelante:

```
Psize = 980; % tamaño de la población
repro = 600; % cantidad de individuos reproductores
maxiter = psize; % numero de iteraciones
pmut = 1/psize; % probabiliad de mutación
l33t = 50; % cantidad de elitismo
emut = 0; % cantidad de élite que NO muta
```

Se carga un grupo de imágenes para hacer un conjunto de pruebas lo más variadas posible:

```
images = cat(3,imread('enblanco.bmp'), imread('ennegro.bmp'),
imread('unaelipse.bmp'), imread('circulogrande.bmp'),
imread('circulograndeychico.bmp'), imread('elipse2.bmp'),
imread('elipse3.bmp'),imread('elipse4.bmp'),imread('elipse5.bmp'),
imread('2elipses.bmp'), imread('circulochico.bmp'),
imread('circulochicorelleno.bmp'), imread('barranegra.bmp'),
imread('cosa1.bmp'), imread('cosa2.bmp'));
images = cat(3,images,
~imread('Assorted_United_States_coins_contraste_alto_Gray_Scale_100x100_sobel_mono.bmp'));
images = cat(3,images, ~imread('Gold-
Coins_alto_contraste_grayscale_100x100_sobel_mono.bmp'));
images = cat(3,images, ~imread('pound-
coins2_contraste_alto_gray_scale_100x100_sobel_mono.bmp'));
I = images(:,:,16); % imagen a usar en esta ejecución
```

Para recuperar la información útil de la población se inicializan dos variables: HoF (hall of fame) y medias. La inicialización de la población se hace mediante la función escrita en clase 'Inicializa_poblacion(psize)' sensiblemente modificada y se prepara la imagen para mostrar los resultados.

```
HoF = zeros(maxiter,30); % Hall of fame (mejores elipses)
medias = zeros(maxiter,1); % vector de medias
%% Inicializacion
Pop = Inicializa_poblacion(psize);
subplot(2,1,2);
imshow(I)
hold on
```

Los siguientes pasos: fitness, selección, cruce y mutación, se hacen en funciones a parte.
El segmento 'Rellenar población' sencillamente sustituye los individuos podados por otros nuevos.
Al final del bucle principal hay una sección de recopilación para mostrar sobre la imagen las mejores elipses de algunas generaciones. Las elipses más antiguas tienen un color azul y las más nuevas tienen color rojo. El color cambia progresivamente.

```
%% BUCLE PRINCIPAL
for iter=1:maxiter
%% Calculo Fitness
    fitn = fitness2(Pop,I);
    p_f = flipud(sortrows([Pop fitn],30)); % Ordenar de mayor a menor
%% Seleccion
    chosen = p_f(1:repro,1:29);
%% Cruce
    chosen(l33t+1:repro,:) = sexyTime(chosen(1:repro-l33t,:));
%% Mutacion :)
    chosen(emut+1:end) = chernobyl(chosen(emut+1:end),pmut);
%% Rellenar población
    Pop(1:repro,:) = chosen(:,:);
    Pop(repro+1:end,:) = Inicializa_poblacion(psize-repro);
%% Recopilación
    HoF(iter,:) = p_f(1,:);
    avrg = mean(p_f,1);
    medias(iter) = avrg(end);
    if(mod(iter,20)==0)
        disp([iter HoF(iter,end) medias(iter)])
        [x,y] = elipseCoordinates(HoF(iter,1:29),I,0:0.1:2*pi);
        plot(x,y,'Color',[iter/maxiter 0 1-(iter/maxiter)])
    end

end
```

Por último se muestran los datos de la ejecución y se resalta la solución definitiva con un color diferente. La función elipseCoordinates devuelve los vectores x e y que forman los puntos de una elipse codificada por el cromosoma que se pasa como parámetro. Tambien se le pasa una imagen y un vector de valores alpha.

```
%% Fin
subplot(2,1,1)
t = 1:maxiter;
%Muestra la evolución del óptimo y de la media
plot(t,HoF(:,end),'-r',t,medias(:,1),'-b');
subplot(2,1,2)
hold on
%Muestra la solucion en verde
[x,y] = elipseCoordinates(HoF(end,1:29),I,1:0.1:2*pi);
plot(x,y,'g')
hold off
```

3.2 Inicialización – Inicializa_poblacion.m

La población se inicializa como una matriz de n por m, donde n es el tamaño de la población y m el tamaño en bits de los cromosomas. La función rand por defecto genera una distribución uniforme entre cero y uno, así que el redondeo dará más o menos la misma probabilidad para ceros y unos.

```
function [Pop]=Inicializa_poblacion(tam)
    Pop = round(rand(tam,29));
end
```

3.3 Función de Fitness – fitness2.m

Para calcular el fitness se utiliza elipseCoordinates y, con los puntos obtenidos, se comprueba si la posición x,y dentro de la imagen es un pixel negro. Para no contar dos veces el mismo pixel, cada cromosoma duplica la imagen y pone en blanco cualquier pixel ya visitado.

```
function [fitn] = fitness2(Pop,I)
    alpha = 0:0.1:2*pi;
    [x,y] = elipseCoordinates(Pop,I,alpha);
    P = size(Pop,1);
    fitn = zeros(P,1);
    for e=1:1:P
        I2 = I;
        for i=1:1:size(x,2)
           % pone cada pixel visitado a 1
           fitn(e) = fitn(e)+~(I2(y(e,i),x(e,i)));
           I2(y(e,i),x(e,i)) = 1;
        end
    end
    pngr = fitn;
    fitn = pngr./size(x,2);  % version implementada actualmente
end
```

3.4 Decodificación de cromosomas - elipseCoordinates.m

La forma en la que se calculan las elipses a partir de unos valores binarios se ha ido complicando poco a poco para poder tener en cuenta elipses diferentes. Por ejemplo, una elipse centrada en 50,50 podrá tener un 'a' de 50 si la elipse es horizontal pero podría seguir cabiendo dentro de la imagen si midiera 65 y estuviera rotada 45º. He cambiado el escalado y ajuste original de las elipses con esperanza de contemplar un numero mayor de elipses posibles, meramente como ejercicio personal. Concretando, el escalado de los valores binarios hacia valores reales se hace de la siguiente manera:

- Se extrae theta, que estará entre algo más de cero y pi/2.
- En función de theta se calcula 'a' de forma que, como máximo y estando centrada, sea capaz de casi tocar los bordes de la imagen en cualquier ángulo.
- En función de 'a' y theta, se calcula 'b' de forma que, como máximo, sea capaz de casi tocar los bordes de la imagen.
- Se escalan x0 e y0 de forma que, como máximo, las elipses rocen los bordes mayores y como mínimo, los inferiores.

La función produce muchos valores de salida que sólo se aprovechan totalmente durante las pruebas. Por lo general sólo interesan 'x' e 'y'. La cantidad de argumentos de entrada también sirve simplemente para dar flexibilidad a las pruebas.

```
function [x,y,a,b,x0,y0,theta] = elipseCoordinates(Pop,I,alpha)
    if nargin < 3
      alpha=0:0.1:2*pi;
    end
    if nargin < 2
      I=ones(100);
    end
    iw = size(I,2)-1;
    ih = size(I,1)-1;
    % Extracción de valores binarios
    bina = binario_a_entero(Pop(:,1:5));
    binb = binario_a_entero(Pop(:,6:10));
```

```
    binx0 = binario_a_entero(Pop(:,11:16));
    biny0 = binario_a_entero(Pop(:,17:22));
    theta = binario_a_entero(Pop(:,23:29));
```

Este segmento escala las variables. El ajuste y escalado de elipses no es la parte interesante ni el objetivo de la práctica y por eso no entraré en muchos detalles.

```
% ESCALADO
    theta = (theta+1)./128;
    theta = theta.*pi./2;
    a = (bina); % eje principal
    a = a./(34-0);
    maxa = (50./max(sin(theta),cos(theta))-5);
    a = floor(a.*maxa);
    a = a+5;
    b = binb; % segundo eje en función del primero
    b = b./(34-0);
    maxb = min((50-a.*sin(theta))./cos(theta),(50-a.*cos(theta))./sin(theta));
    maxb = min(maxb,(50+a.*sin(theta))./cos(theta));
    maxb = min(maxb,(50+a.*cos(theta))./sin(theta));
    b = floor(b.*maxb);

    %% Ajuste de posicion
    x0max = (a.*cos(theta)+b.*sin(theta));
    x0min = (iw-x0max);
    y0max = (a.*sin(theta)+b.*cos(theta));
    y0min = (ih-y0max);
    x0 = binx0/70;
    x0 = x0.*(x0max-x0min)+x0min;
    y0 = biny0/70;
    y0 = y0.*(y0max-y0min)+y0min;
```

Como las operaciones se hacen matricialmente y ahora hace falta sumar un vector columna a cada columna de una matriz, la solución es multiplicar horizontalmente los valores del vector columna. Por eso, se multiplican x0, y0 y theta por un vector horizontal de unos.
Como resultado, 'x' e 'y' son matrices con tantas filas como individuos de la población, y tantas columnas como valores de 'alpha'.

```

    %% Expansión horizontal de x0,y0 y theta para poder sumarlos a la matriz
    unos = ones(1,size(alpha,2));
    x0 = x0*unos;
    y0 = y0*unos;
    theta = theta*unos;

    %% calculo de puntos
    x = round((a*cos(alpha)).*cos(theta) - (b*sin(alpha)).*sin(theta) +x0);
    y = round((a*cos(alpha)).*sin(theta) + (b*sin(alpha)).*cos(theta) +y0);
end
```

3.5 Decodificación de genes - binario_a_entero.m

Toma una matriz de ceros y unos, donde cada fila es un número codificado en binario. Devuelve la columna de valores transformados a entero.

```
function [entero] = binario_a_entero(binario)
    entero = zeros(size(binario,1),1);
    L = size(binario,2);
    for i=1:L
        entero(:) = entero(:) + binario(:,i)*2^(i-1);
    end
end
```

3.6 Cruce - sexyTime.m

Se calculan los puntos de cruce de tal forma que se garantiza que cada hijo tiene tres segmentos de dos padres diferentes. El primer punto debe ser al menos 2, el segundo punto debe ser al menos 2 posiciones superior al primero y a la vez menor que la última posición.
El segmento del centro se intercambia entre los cromosomas pares e impares de una reordenación aleatoria de los padres.

```
function [prole] = sexyTime(padres)
    prole = padres(randperm(size(padres,1)),:);  %Parejas aleatorias
    minwindow = 1;
    max_primer = size(padres,2)-(2+minwindow);
    max_segundo = size(padres,2)-1;
    for i = 1:2:numparejas  % de dos en dos
        punto1 = round(rand().*(max_primer-2)+2);
        punto2 = round(rand().*(max_segundo-(punto1+2))+punto1+2);
        prole(i,punto1:punto2) = padres(i+1,punto1:punto2);
        prole(i+1,punto1:punto2) = padres(i,punto1:punto2);
    end
end
```

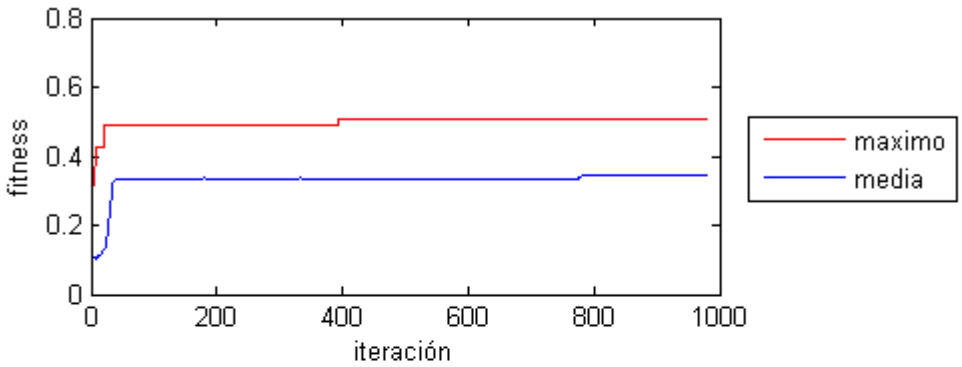
3.7 Mutación - chernobyl.m

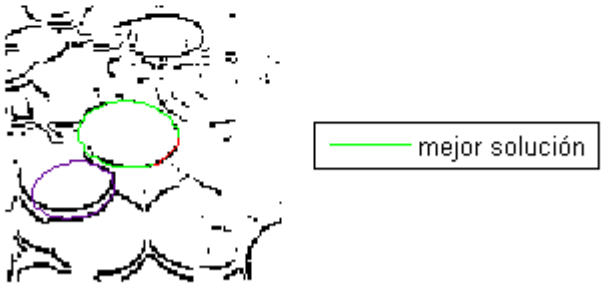
Los elegidos es el fragmento de la población que no se ha podado. Por cada uno se decide si debe mutar o no y luego qué bit debe ser invertido.

```
function [mutated_chosen] = chernobyl(chosen,pmut)
    mutated_chosen = chosen;
    for i = 1:size(chosen,1)
        if rand(1)>= (1-pmut)

            binary_index = floor((rand().*29)+1);
            mutated_chosen(i,binary_index) = ~mutated_chosen(i,binary_index);
        end
    end
end
```

El resultado de la ejecución del código tal cual está mostrado aquí podría tener esta forma:





Resultado de una ejecución con unos parámetros que causan una convergencia muy rápida.

4. Ajuste de parámetros

Para iterar a través de varios valores de las variables y, con suerte, poder ver cuales son los mejores, utilizaré un pequeño script que hará el trabajo sucio. Los valores obtenidos serán en función de una sola imagen de prueba (una de monedas, casi siempre) y con los demás parámetros previamente fijados. A continuación se muestra el código para iterar por el tamaño de la población de 20 en 20.

```
fs = [];
ps = [];
for i = 100:20:800
    psize=i
    for j=1:5 % Incrementa este numero para más consistencia
        SGA1
        fs = [fs HoF(end,end)]; %Fitness
        ps = [ps i]; %Valor de la variable de interés
    end
end
figure
plot(ps,fs)
```

4.1 Función de fitness

La función de fitness es la parte más delicada, ya que siempre puede haber duda sobre qué es una buena solución. Incluso en un caso aparentemente tan simple como encontrar elipses surgen preguntas como ¿Es mejor una circunferencia que un círculo?¿En una imagen que sea totalmente del color objetivo (negro en mi caso) cualquier solución se debería considerar óptima?¿Y si sale una elipse casi plana o muy pequeña?
Para responder a estas preguntas podría preguntarme cómo resolvería yo, con mis ojos, el problema pero, al final, hacer demasiado sofisticada una respuesta le quita encanto a la simplicidad que se pretende buscar en el algoritmo genético simple.

Por lo anterior, todas las funciones de fitness que he considerado, recorren una sola vez cada punto de la elipse candidata, contándose la cantidad de píxeles diferentes que coinciden con el color objetivo. La diferencia entre las funciones de fitness probadas son simplemente variaciones en la relación entre los puntos contados, puntos totales y radios de la elipse.

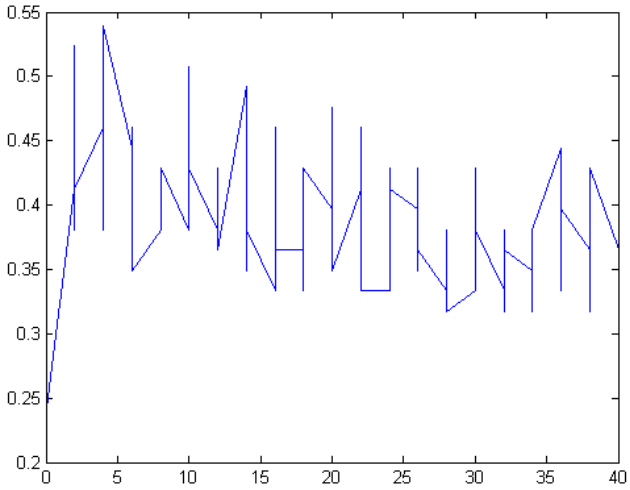
Con una población de 200, número de iteraciones de 200, probabilidad de mutación constante en la inversa de la población, poda al 30% aproximadamente y un elitismo de 2 individuos, he iterado entre varias funciones de fitness y más o menos todas convergen hacia una solución aceptable.

La solución que se muestra en el código de antes es la más segunda más simple de todas las que parecían suficientemente buenas. Las demás proporcionan soluciones muy similares pero con rangos de valores diferentes.
La que es todavía más simple que ésta, consistía en hacer un escalado entre 0 y 1 del número de pixeles que hacen match. En la función elegida se hace un escalado, pero contando con el número de puntos de la elipse, de modo que se favorecen elipses más grandes independientemente de su excentricidad.
Funcion de fitness:

```
fitn = pngr./size(x,2);
```

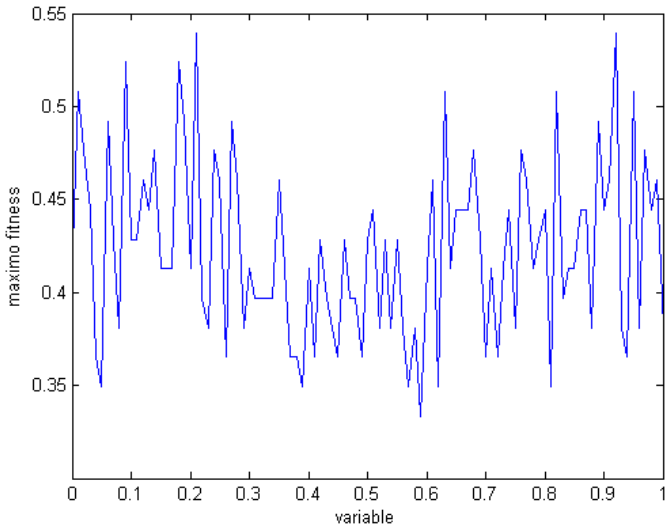
4.2 Elitismo y truncamiento

Independientemente del valor de cualquier otra variable durante las pruebas realizadas, la presencia de elitismo, aunque sea de 2 frente a una población de millares, da mucha uniformidad a la salida del script. Los miembros élite son vulnerables a la mutación, pero es muy poco probable que una mutación nociva sea perjudicial para la evolución en una iteración dada.



Las pruebas indican que una presencia de entre 2 y 4 miembros élites es lo óptimo.

El criterio para seleccionar la poda también es muy sencillo. Elegiremos una cantidad de individuos reproductores en función de de la población máxima: un porcentaje. El algoritmo es más rápido cuantos más individuos se eliminen, por eso de que es más rápido generarlos desde cero que reproducirlos, así que utilizaré el porcentaje de poda más alto posible (o porcentaje de reproductores más bajo) que no reduzca mucho la eficacia del algoritmo.
Haciendo varias ejecuciones con diferentes iteraciones por cada valor de la variable, la tendencia es que haya fitness medios bajos más o menos por la zona de la mitad del rango de valores del porcentaje de individuos reproductores.



Este es el resultado de probar 100 valores distribuidos uniformemente entre 0 y el tamaño de la población. Se muestra la media de 40 ejecuciones por cada valor.

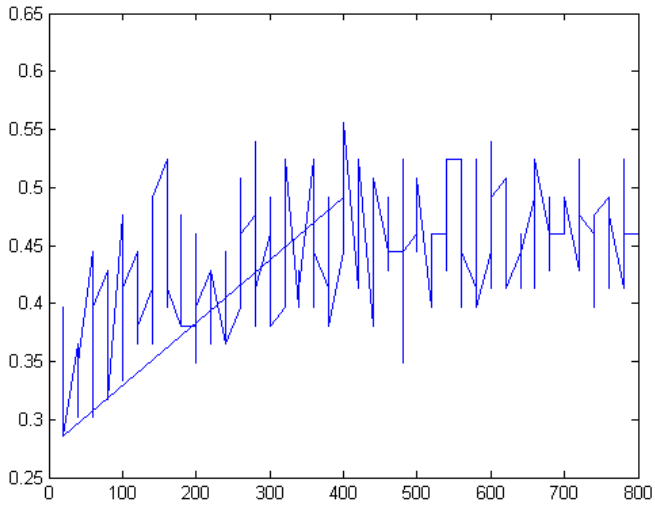
Tras las 4000 iteraciones, no parece seguro, pero sí al menos probable, que a partir del 20% de individuos reproductores no se gane fitness medio.

Por eso, creo que un valor de 20% es apropiado.

4.3 Numero de iteraciones y tamaño de la población

Al igual que para calcular el truncamiento, disponemos las otras variables con valores estáticos e iteramos a través del parámetro de interés. Comprobamos que con una población de 800 individuos ya se alcanza con facilidad el fitness máximo alrededor de las 120 iteraciones máximas.

Poniendo esas mismas 120 iteraciones máximas y el resto de los valores calculados, iteramos ahora sobre el tamaño de la población. De 20 a 800 , de veinte en veinte, 5 iteraciones cada vez, se dibuja una curva que sugiere unos valores entre 100 y 300. Concretamente a partir de 160 no se mejora mucho más.
Así que elijo 160 como valor para el tamaño de la población.



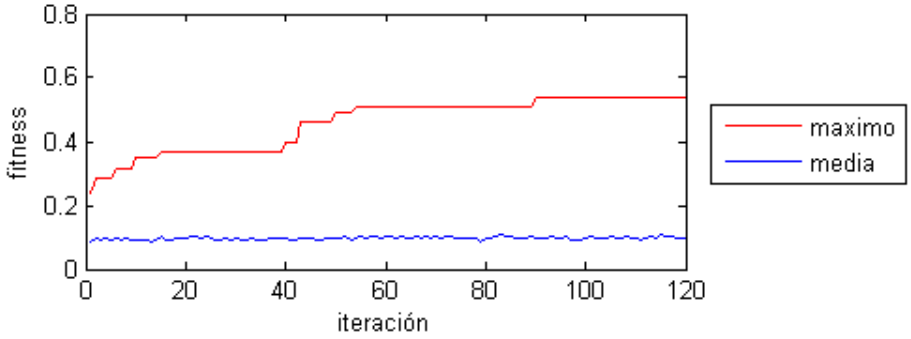
4.4 Probabilidad de mutación

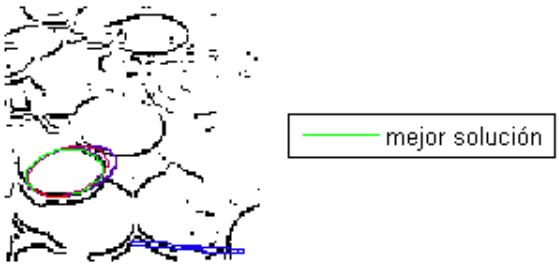
Siguiendo la misma dinámica que previamente, itero por diferentes valores de la probabilidad de mutación. Parece que a partir de un porcentaje muy bajo, en torno al 1% las posibles mejoras no son significativas, lo que apunta hacia la sospecha de que una probabilidad de mutación inversamente proporcional al tamaño de la población es una cantidad válida.

5. Resultado y conclusión

Finalmente, los hiperparámetros quedan ajustados con los siguientes valores:

- Tamaño de la población: 160
- Número de iteraciones: 120
- Probabilidad de mutación: 1/|poblacion|
- Elitismo: 2
- Poda: 80% |población|





Pues ya tenemos el algoritmo genético ajustado. Hay que notar que tener unas variables realmente optimizadas implicaría explorar un espacio de tantas dimensiones como parámetros, un trabajo difícil que se supone que estamos intentando evitar al aplicar un algoritmo genético en primer lugar. ¿Podríamos usar un SGA para encontrar los parámetros óptimos? No sé, prefiero usar los valores obtenidos previamente; al fin y al cabo, son suficientemente buenos.

5. Código comentado

SGA1.m

```
%% Meta parámetros
% Tamaño de la población
psize = 160;
% Reproductores por generacion (psize-poda)
repro = 2*round((psize/10)); %aproximadamente un 20% y que sea par
% Numero de iteraciones
maxiter = 120;
% Probabilidad de mutación
pmut = 1/psize;
% Cantidad de elitismo
l33t = 2;
% Miembros de la elite que NO pueden mutar
emut = 0;
images = cat(3,imread('enblanco.bmp'), imread('ennegro.bmp'),
imread('unaelipse.bmp'), imread('circulogrande.bmp'),
imread('circulograndeychico.bmp'), imread('elipse2.bmp'),
imread('elipse3.bmp'),imread('elipse4.bmp'),imread('elipse5.bmp'),
imread('2elipses.bmp'), imread('circulochico.bmp'),
imread('circulochicorelleno.bmp'), imread('barranegra.bmp'),
imread('cosa1.bmp'), imread('cosa2.bmp'));
images = cat(3,images,
~imread('Assorted_United_States_coins_contraste_alto_Gray_Scale_100x100_sobel_m
ono.bmp'));
images = cat(3,images, ~imread('Gold-
Coins_alto_contraste_grayscale_100x100_sobel_mono.bmp'));
images = cat(3,images, ~imread('pound-
coins2_contraste_alto_gray_scale_100x100_sobel_mono.bmp'));
I = images(:,:,16); % imagen a usar en esta ejecución
% Hall of fame (mejores elipses OF ALL TIME)
HoF = zeros(maxiter,30);
% Fitness de media
medias = zeros(maxiter,1);

%% Inicializacion
Pop = Inicializa_poblacion(psize);

subplot(2,1,2);
imshow(I)
hold on
%% BUCLE PRINCIPAL
for iter=1:maxiter
%% Calculo Fitness
    %Fitness esta calculado de forma que representa una proporcion
    %de pixeles que son cubiertos por la elipse (0. .. 1.)
    fitn = fitness2(Pop,I);
    % Ordenar de mayor a menor por el fitness
    p_f = flipud(sortrows([Pop fitn],30));
%% Seleccion
    chosen = p_f(1:repro,1:29);
%% Cruce
    chosen(l33t+1:repro,:) = sexyTime(chosen(1:repro-l33t,:));
%% Mutacion :)
    chosen(emut+1:end) = chernobyl(chosen(emut+1:end),pmut);
%% Rellenar población, prepara la siguiente iteracion
    Pop(1:repro,:) = chosen(:,:);
    Pop(repro+1:end,:) = Inicializa_poblacion(psize-repro);
%% Recopilación
    HoF(iter,:) = p_f(1,:);
    avrg = mean(p_f,1);
```

```
    medias(iter) = avrg(end);
    if(mod(iter,round(maxiter/20))==0)
        disp([iter HoF(iter,end) medias(iter)])
        [x,y] = elipseCoordinates(HoF(iter,1:29),I,0:0.1:2*pi);
        plot(x,y,'Color',[iter/maxiter 0 1-(iter/maxiter)])
    end
end
%% Fin
subplot(2,1,1)
t = 1:maxiter;
%Muestra la convergencia a la solucion final
axis equal
plot(t,HoF(:,end),'-r',t,medias(:,1),'-b');
legend('maximo','media','Location','EastOutside');
ylabel('fitness')
xlabel('iteración')
subplot(2,1,2)
hold on
%Muestra la solucion en verde
[x,y] = elipseCoordinates(HoF(end,1:29),I,1:0.1:2*pi);
e=plot(x,y,'g');
legend(e,'mejor solución','Location','EastOutside');
hold off
```

Inicializa_poblacion.m

```
function [Pop]=Inicializa_poblacion(tam)
    Pop = round(rand(tam,29));
end
```

fitness2.m

```
function [fitn] = fitness2(Pop,I)
    alpha = 0:0.1:2*pi;
    [x,y] = elipseCoordinates(Pop,I,alpha);
    % calculo fitness
    %comprobamos el matching con la imagen
    P = size(Pop,1);
    fitn = zeros(P,1);
    for e=1:1:P
        I2 = I;
        for i=1:1:size(x,2)
           % pone cada pixel visitado a 1
           fitn(e) = fitn(e)+~(I2(y(e,i),x(e,i)));
           I2(y(e,i),x(e,i)) = 1;
        end
    end
    pngr = fitn;
    % Nummero de puntos negros, en lugar de blancos
    %% Funciones de fitness: Por defecto saca nº de pixeles (que no está tan
mal)
    %% [0] cuantos más mejor
%     fitn = 1-1./(pngr+1);
    %% [1] Proporcion de puntos negros / puntos de la elipse
    % El que mejor resultados ha dado
    fitn = pngr./size(x,2);
    %% [2] cociente de summas (presentes entre mmaximos)
%     fitn = (pngr+a+b)./(size(x,2)+36+36);
    %% [3] Potenciar color-matches
%     fitn = 1-1./(pngr+1);
%     fitn = fitn.*sqrt(a.*b)/36;
    %% [4] Potenciar baja excentricidad
%     fitn = pngr/size(x,2).*sqrt(2./((a./b)+(b./a)));
```

```
    %% [5] Cantidad de puntos negros y proporción de ptos negros
%     fitn = pngr.*(a+b)/(72*63);
    %% [6] Cantidad de ptos, proporción de ptos y baja excentricidad
%     fitn = pngr.*(a+b).*2./(72*63.*((a./b)+(b./a)));
    %% [7] A lo bestia. ptos/ptos_POSIBLES
%     fitn = pngr./(2*size(I,2));
end
```

ElipseCoordinates.m

```
function [x,y,a,b,x0,y0,theta] = elipseCoordinates(Pop,I,alpha)
    if nargin < 3
      alpha=0:0.1:2*pi;
    end
    if nargin < 2
      I=ones(100);
    end
    iw = size(I,2)-1;
    ih = size(I,1)-1;
    % Extracción de valores binarios
    bina = binario_a_entero(Pop(:,1:5));
    binb = binario_a_entero(Pop(:,6:10));
    binx0 = binario_a_entero(Pop(:,11:16));
    biny0 = binario_a_entero(Pop(:,17:22));
    theta = binario_a_entero(Pop(:,23:29));

    % ESCALADO
    % A[a..b] -> C[c..d]
    % C = (A-a)*(d-c)/(b-a)+c
    % theta[0..127] -> theta[0..pi/2]
    theta = (theta+1)./128;
    theta = theta.*pi./2;
    % 5<=a<=36;
    % a[0..31] -> a[0..49]
    % a = a*(49-5)/31+5
    % 19<=x0<=82
    % x0[0..63] -> x0[5..95]
    % x0 = (x0)*(90)/63+5;

    %% [1] ajuste chano
%     a = round(bina+5);
%     b = round(binb+5);
%     x0 = round(binx0+19);
%     y0 = round(biny0+19);

    %% [2] ajuste tocho
    % eje principal
    a = (bina);
    a = a./(34-0);
    maxa = (50./max(sin(theta),cos(theta))-5);
    a = floor(a.*maxa);
    a = a+5;
%     b = 0*binb;
    % segundo eje en funcion del primero
    b = binb;
    b = b./(34-0);
    % dada una a, bmax será igual a la mitad del ancho del rectángulo máximo
    % bmax = min(proyecciones perpendiculares a a sobre los bordes de la
imagen)
    % hay cuatro variaciones de la proyección (4 bordes de la imagen)
    maxb = min((50-a.*sin(theta))./cos(theta),(50-a.*cos(theta))./sin(theta));
    maxb = min(maxb,(50+a.*sin(theta))./cos(theta));
    maxb = min(maxb,(50+a.*cos(theta))./sin(theta));
```

```
    b = floor(b.*maxb);
    %% Ajuste de posicion
    x0max = (a.*cos(theta)+b.*sin(theta));
    x0min = (iw-x0max);
    y0max = (a.*sin(theta)+b.*cos(theta));
    y0min = (ih-y0max);
    x0 = binx0/70;
    x0 = x0.*(x0max-x0min)+x0min;
    y0 = biny0/70;
    y0 = y0.*(y0max-y0min)+y0min;

    %% Expansión horizontal de x0,y0 y theta para poder sumarlos a la matriz
    unos = ones(1,size(alpha,2));
    x0 = x0*unos;
    y0 = y0*unos;
    theta = theta*unos;

    %% calculo de puntos
    x = round((a*cos(alpha)).*cos(theta) - (b*sin(alpha)).*sin(theta) +x0);
    y = round((a*cos(alpha)).*sin(theta) + (b*sin(alpha)).*cos(theta) +y0);
%     x = round((a*cos(alpha)+x0).*cos(theta) + (b*sin(alpha)+y0).*sin(theta));
%     y = round((a*cos(alpha)+x0).*sin(theta) + (b*sin(alpha)+y0).*cos(theta));
end
```

binario_a_entero.m

```
function [entero] = binario_a_entero(binario)
    entero = zeros(size(binario,1),1);
    L = size(binario,2);
    for i=1:L
        entero(:) = entero(:) + binario(:,i)*2^(i-1);
    end
end
```

sexyTime.m

```
function [prole,punto1,punto2] = sexyTime(padres)
    prole = padres(randperm(size(padres,1)),:); %Parejas aleatorias
    %% generación de puntos de corte
    % versión implementada:
    % 1 [->] Un par de puntos de corte por cada par de padres
    % 2 [  ] El mismo par de puntos de corte para toda la generación
    minwindow = 1;
    max_primer = size(padres,2)-(2+minwindow);
    max_segundo = size(padres,2)-1;
     %% Versión 2
%     punto1 = round(rand().*(max_primer-2)+2);
%     punto2 = round(rand().*(max_segundo-(punto1+2))+punto1+2);
%     prole(1:2:end,punto1:punto2) = prole(2:2:end,punto1:punto2);
%     prole(2:2:end,punto1:punto2) = prole(1:2:end,punto1:punto2);
%     punto2 = randi([punto1+2 max_segundo]);
    %% Versión 1
    for i = 1:2:size(padres,1) % de dos en dos
        punto1 = round(rand().*(max_primer-2)+2);
        punto2 = round(rand().*(max_segundo-(punto1+2))+punto1+2);
        prole(i,punto1:punto2) = padres(i+1,punto1:punto2);
        prole(i+1,punto1:punto2) = padres(i,punto1:punto2);
    end
end
```

chernobyl.m

```
function [mutated_chosen] = chernobyl(chosen,pmut)
```

```
    mutated_chosen = chosen;
    for i = 1:size(chosen,1)
        %Una forma más de tener una dist. uniforme
        if rand(1)<=pmut
%             binary_index = randi([1 29]);
            binary_index = floor((rand().*29)+1);
            mutated_chosen(i,binary_index) = ~mutated_chosen(i,binary_index);
        end
    end
end
```

PruebaPop.m (muestra datos estadísticos sobre las elipses que se generan)

```
% prueba pop
Pop = Inicializa_poblacion(2000);
[x,y,a,b,x0,y0,theta] = elipseCoordinates(Pop);
subplot(4,1,1)
grid on
axis([-10 110 -10 110])
axis equal
hold on
for i=1:1:size(x,1)
%     if ((min(x(i,:))<1) | (min(y(i,:))<1))
%         disp([(min(x(i,:))),(min(y(i,:)))]);
%     end
%     if ((max(x(i,:))>100) | (max(y(i,:))>100))
%         disp([(max(x(i,:))),(max(y(i,:)))]);
%     end
    plot( x(i,:),y(i,:));
end
hold off
subplot(4,1,2)
[c,b] = hist(a);
plot(b,c)
title('a')
subplot(4,1,3)
[c,b] = hist(x0);
plot(b,c)
title('x0')
subplot(4,1,4)
[c,b] = hist(theta);
plot(b,c)
title('theta')
% plot(sort(50.*(cos(theta)+sin(theta))))
```

iteraParametro.m (conviene comentar en SGA1.m la variable de interés)

```
fs = [];
ps = 0:0.005:0.1;
for i = ps
    pmut = i
    for j=1:100 % Incrementa este numero para más consistencia
        fval = HoF(end,end);
        SGA1
    end
    fs = [fs mean(fval)];
end
figure
plot(ps,fs)
xlabel('variable')
ylabel('maximo fitness')
```