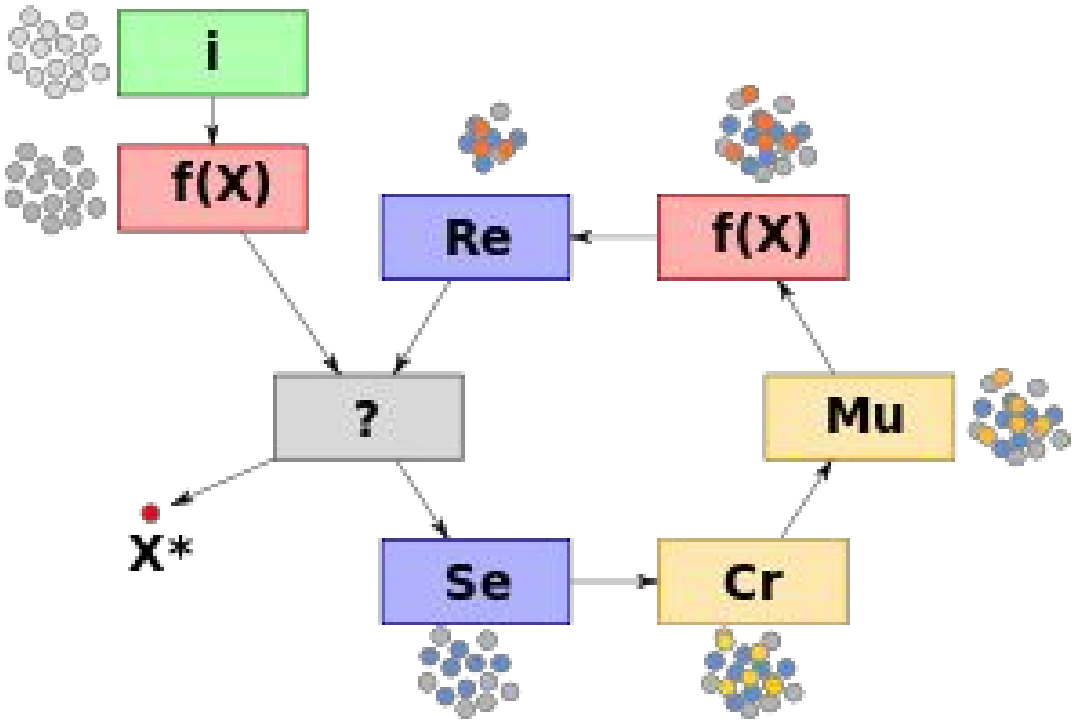
Bloque 1.
Algoritmos Genéticos.



Manuel Montero Santana.

Sistemas Inteligentes 1. Grado en Ingeniería Informática. Escuela de Ingeniería Informática Curso 2015/2016

Introducción:

El objeto de esta práctica es la adquisición de los conocimientos mínimos para con un algoritmo genético sencillo (SGA) resolver un problema de optimización combinatoria no determinística. Cuando los problemas combinatorios son excesivamente complejos, con espacios de búsqueda inabordables por métodos analíticos ni fuerza bruta, se hace preciso el uso de técnicas de búsqueda "a ciegas". Una de las más populares, con origen en la teoría de la evolución de Charles Darwin, que tiene utilidad en multitud de casos son los Algoritmos Genéticos.

Podemos resumir en que un algoritmo genético sencillo podría programarse siguiendo el siguiente pseudocódigo:

1. Inicializar la población
2. Cálculo del Fitness de cada individuo
3. Selección
4. Cruce
5. Mutación
6. Mientras Evaluación de la condición de finalización sea falso, volver a 2.

Desarrollo de la práctica:

El problema que debemos resolver en esta práctica es la búsqueda de elipses en imágenes. Para ello utilizaremos una **codificación de cromosomas** binaria.

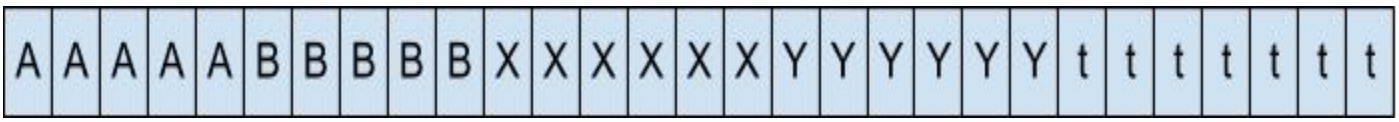
Para representar una elipse necesitamos 5 variables:

Tamaño del semieje mayor (A).

Tamaño del semieje menor (B).

Dos coordenadas para representarlo en el plano (X e Y).

Una variable Theta para rotar la elipse.



La **función de fitness** para evaluar los cromosomas se basa en dibujar la elipse encima de la imagen de muestra y comprobar la cantidad de puntos de color negro que coinciden con supuestas elipses.

Tras aplicar la función de fitness a toda la población debemos aplicar los operadores de **selección cruce y mutación**. La **selección** se basa simplemente en determinar qué cantidad de individuos es considerada buena para seguir generando nuevos individuos. El **cruce** tiene dos partes, elitismo y truncado. El elitismo consiste en reproducir los dos mejores individuos de la población actual a la nueva población. Una vez hecho esto, con los individuos seleccionados tras la operación de selección se generan todos lo nuevos individuos de la nueva población. Para generar un par de individuos basta con seleccionar un padre y una madre para acabar dividiendolos de manera aleatoria en tres partes para luego mezclarlas y generar los nuevos individuos.

Tras esta operación se someten todos los individuos a la **mutación**. La mutación consiste en alterar un determinado gen sí a ese individuo "le toca" de forma probabilista.

Este proceso se repite hasta que se alcanza un número de generaciones.

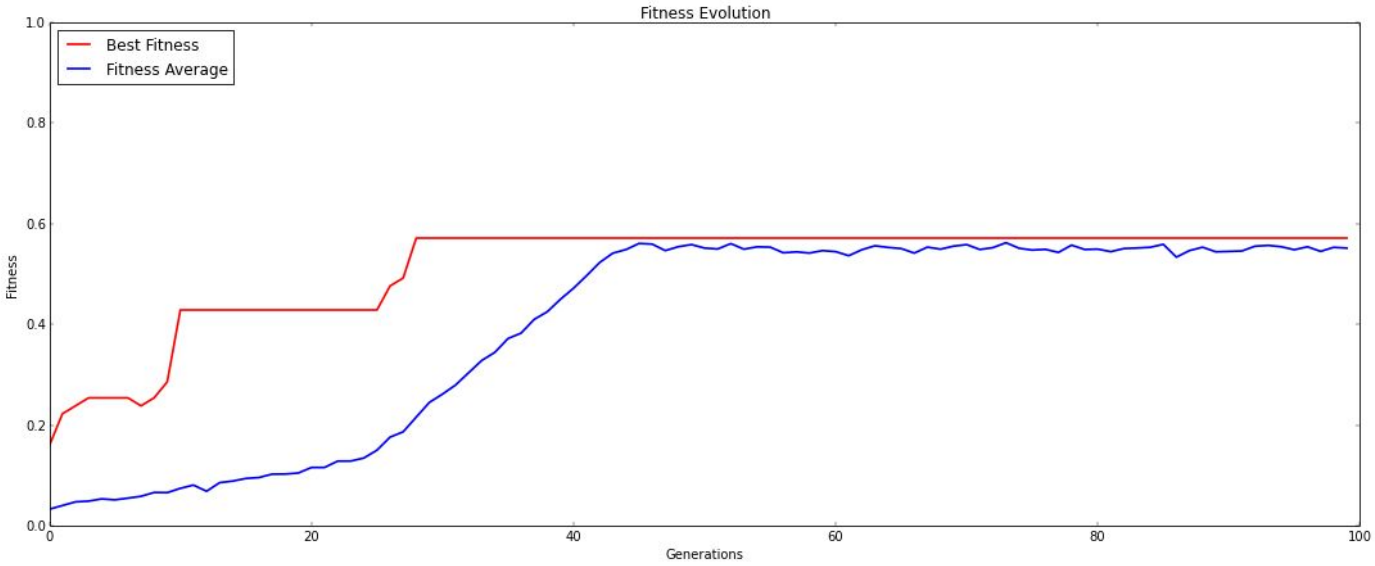
Pruebas:

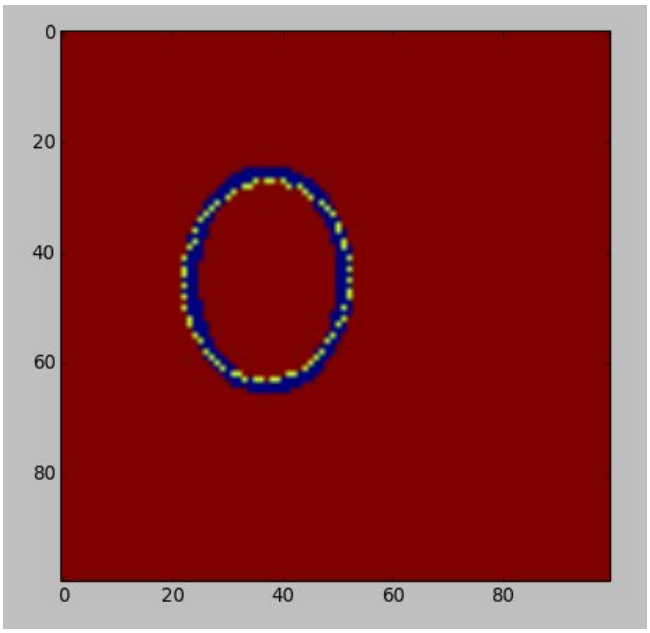
Tras probar con diferentes elipses sencillas y medir resultados determine que unos parámetros aceptables para nuestro problema son:

- Probabilidad de mutación: 15%
- Número de generaciones: 100
- Número de individuos seleccionados: 125
- Número de individuos: 200

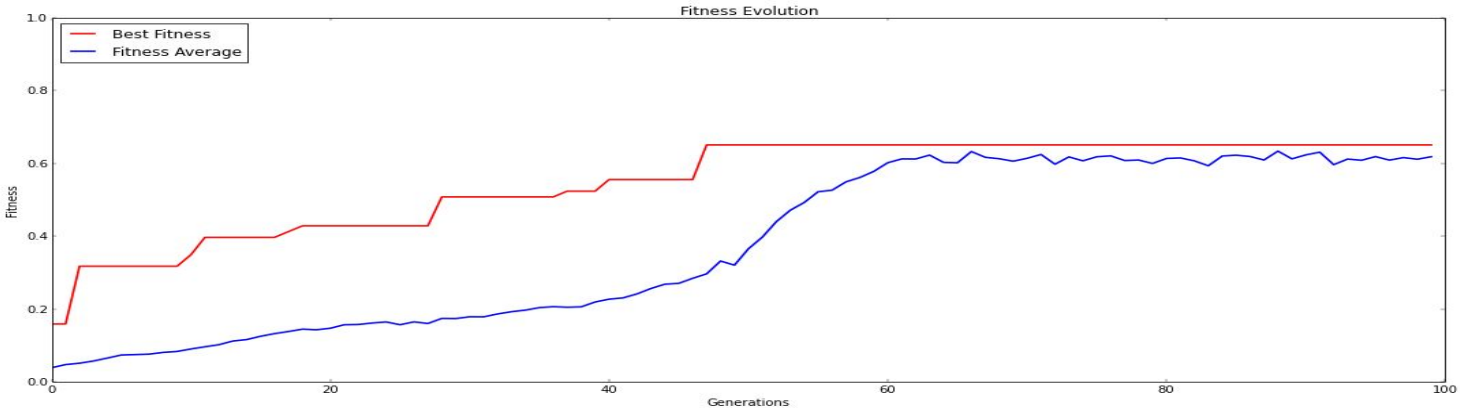
A continuación muestro cuatro ejecuciones con diferentes elipses, donde se puede observar la evolución del fitness y una imagen donde aparece la elipse original (Azul) y la calculada (Amarilla).

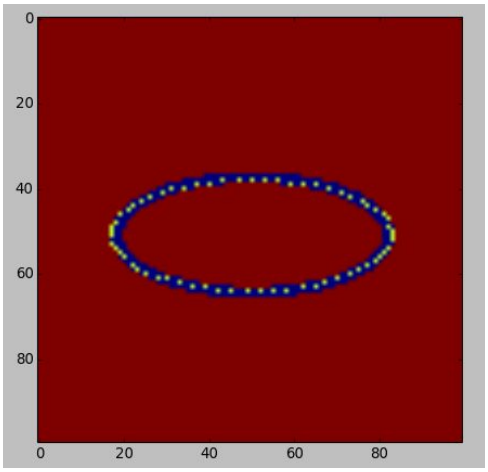
Elipse 1:



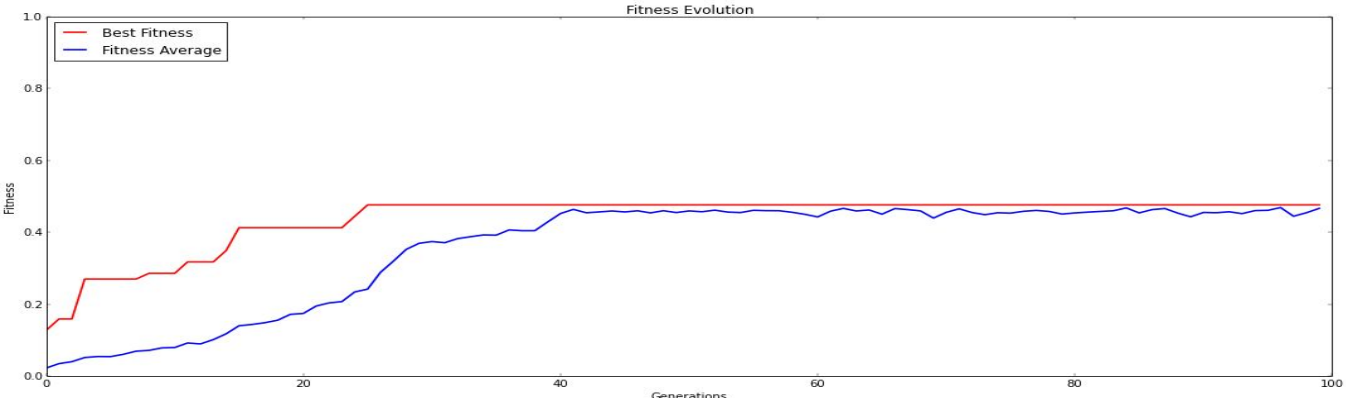


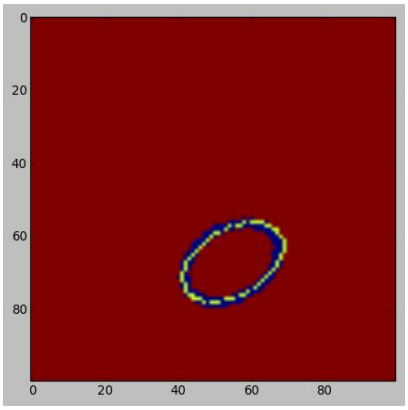
Elipse 2:





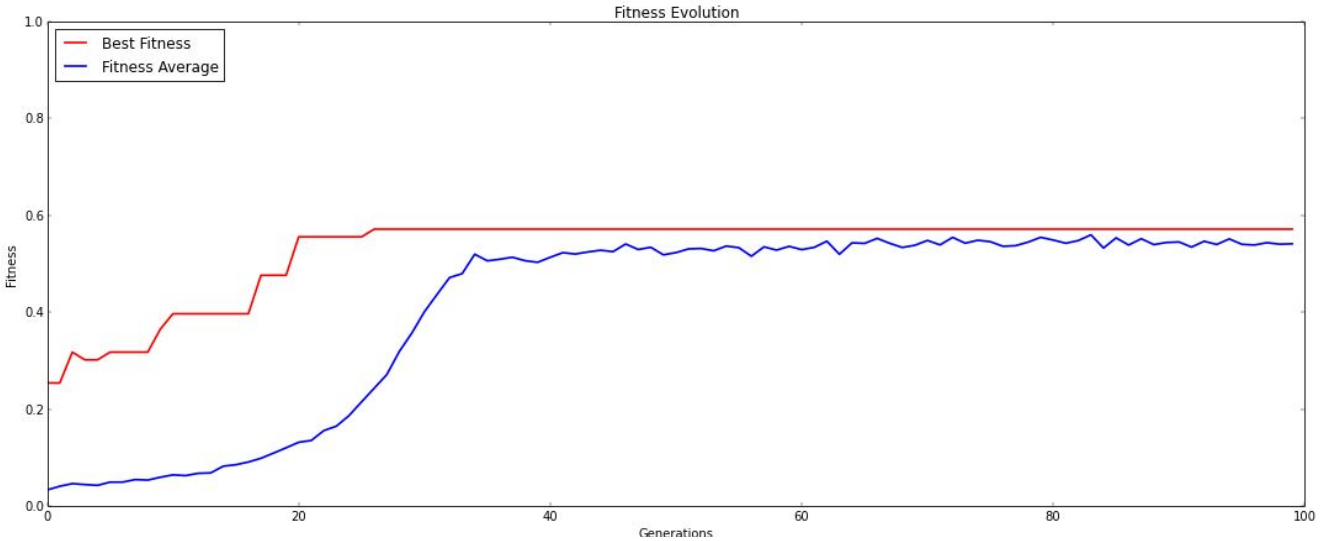
Elipse 3:

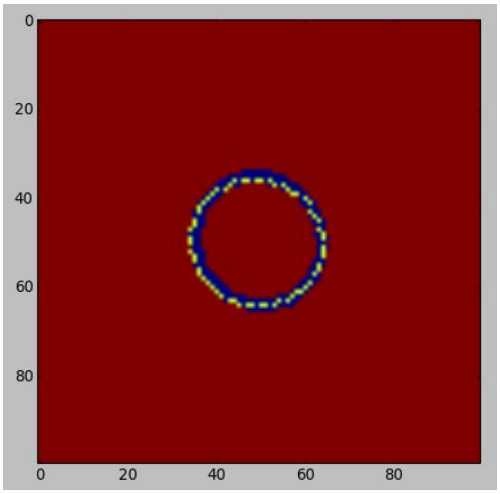




Manuel Montero Santana.

Elipse 4:


Fitness Evolution
Best Fitness
Fitness Average
Fitness
Generations
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


0
20
40
60
80
0
20
40
60
80

Anexo (Código python).

Binario a decimal.

```python
def bin2dec(vector):
    integ = 0
    for i in range(len(vector)):
        integ = integ + vector[i]*2**i
    return integ
```

Herramientas para tratar imágenes.

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

def transformSquareImage(fileName):
    image = Image.open(fileName)
    imagenbw = image.convert("L")
    imagenbw.thumbnail((100, 100), Image.ANTIALIAS)
    imagenbw.save("sample.png")

def image2Matrix(fileName):
    m = np.asarray(Image.open(fileName).convert('L'))
    r = np.zeros((100,100))
    for i in range(m.shape[0]):
        for j in range(m.shape[1]):
            r[i][j]= 255 if m[i][j] > 50 else 0
    return r

def Matrix2Image(matrix):
    plt.imshow(matrix)
    plt.savefig('predict.png')
```

Algoritmo SelectionSort de ordenación.

```python
def selectionsort(fitnessValues,population):

    tam = len(fitnessValues)
    for i in range(0,tam-1):
        max=i
        for j in range(i+1,tam):
            if fitnessValues[max] < fitnessValues[j]:
                max=j

        aux = fitnessValues[max]
        fitnessValues[max] = fitnessValues[i]
        fitnessValues[i] = aux

        for k in range(len(population[0])-1):
            aux2=population[max][k]
            population[max,k] = population[i,k]
            population[i,k] = aux2

    return fitnessValues,population
```

Funciones del algoritmo genético.

```python
import ...

def initialize_population(size):
    population = np.random.rand(*(size,29))
    for i in range(population.shape[0]):
        for j in range(population.shape[1]):
            population[i,j]=round(population[i,j])
    return population

def getEllipse(individual):
    a = Bin2Dec.bin2dec(individual[0:5]) + 5
    b = Bin2Dec.bin2dec(individual[5:10]) + 5
    x0 = Bin2Dec.bin2dec(individual[10:16]) + 19
    y0 = Bin2Dec.bin2dec(individual[16:22]) + 19
    theta = Bin2Dec.bin2dec(individual[22:29])
    theta *= 179/127
    theta *= 3*math.pi/180

    # Create the representation of ellipse
    ellipse = []
    alpha=0
    while alpha <= 2 * math.pi:
        x = round((a*math.cos(alpha)*math.cos(theta) - b*math.sin(alpha)*math.sin(theta)))+x0
        y = round((a*math.cos(alpha)*math.sin(theta) + b*math.sin(alpha)*math.cos(theta)))+y0
        x = x if (x>-1 and x<=99) else 0
        y = y if (y>-1 and y<=99) else 0
        ellipse.append([x,y])
        alpha+=0.1

    bestEllipse = []
    bestEllipse.append([ellipse[0][0],ellipse[0][1]])
    for i in range(1,len(ellipse)):
        if (ellipse[i][0] != ellipse[i-1][0] and ellipse[i][1] != ellipse[i-1][1]):
            bestEllipse.append([ellipse[i][0],ellipse[i][1]])
    return bestEllipse,ellipse,len(ellipse)
```

```python
def fitness(population,image):
    fitnessValues = np.zeros(population.shape[0])
    for i in range(population.shape[0]):
        ellipse,ellipse2,nPoints = getEllipse(population[i])
        # Matching with image
        for j in range(len(ellipse)):
            if image[ellipse[j][0],ellipse[j][1]] < 50:
                fitnessValues[i] += 1
        fitnessValues[i] /= nPoints
    return fitnessValues

def selection(population, numberOfSelectedIndividuals=50):
    population = population[0:numberOfSelectedIndividuals,:]
    return population
```

Manuel Montero Santana.

```
def crossover(population, populationSize):
    newPopulation = np.zeros((populationSize,29))
    newPopulation[0] = population[0]
    newPopulation[1] = population[1]

    for i in xrange(2,populationSize,2):
        father = population[random.randint(0, len(population)-1)]
        mother = population[random.randint(0, len(population)-1)]
        cut1 = random.randint(0, 28)
        cut2 = random.randint(0, 28)
        if (cut1<cut2): smallindex=cut1; biggerindex=cut2
        else: smallindex=cut2; biggerindex=cut1;

        #Crossover
        newPopulation[i,0:smallindex]=father[0:smallindex]
        newPopulation[i,smallindex:biggerindex]=mother[smallindex:biggerindex]
        newPopulation[i,biggerindex:28]=father[biggerindex:28]
        newPopulation[i+1,0:smallindex]=mother[0:smallindex]
        newPopulation[i+1,smallindex:biggerindex]=father[smallindex:biggerindex]
        newPopulation[i+1,biggerindex:28]=mother[biggerindex:28]

    return newPopulation

def mutation(population, mutationPercentage=5):
    for i in range(2,len(population)):
        if (random.randint(0, 100) <= mutationPercentage):
            gene = random.randint(0, 28)
            value = population[i,gene]
            population[i,gene] = 0 if value==1 else 0

    return population
```

Main.

```
mutationPercentage = 15
populationSize = 200
numberOfSelectedIndividuals = 125
generations=100

I.transformSquareImage('4.png')
m = I.image2Matrix('sample.png')
population = GA.initialize_population(populationSize)
for i in range(generations):
    print "Generation: "+ str(i)
    fitnessvalues = GA.fitness(population,m)
    fitnessvalues,population = Ordenation.selectionsort(fitnessvalues,population)
    print "     Best fitness: "+str(fitnessvalues[0])
    print "     Fitness average: "+str(sum(fitnessvalues)/len(fitnessvalues))
    population = GA.selection(population,numberOfSelectedIndividuals)
    population = GA.crossover(population,populationSize)
    population = GA.mutation(population,mutationPercentage)
```