



Instituto Tecnológico de Buenos Aires

Sistemas de Inteligencia Artificial

Trabajo Práctico 2

Algoritmos Genéticos

Ejercicio 1

Pensar (**no es necesario implementar**) cómo implementarían, mediante Algoritmos Genéticos, un programa que tome una imagen cuadrada y trate de representar de la mejor manera posible dicha imagen en un mapa de NxN caracteres ASCII, por ejemplo como se demuestra en la siguiente página: <http://www.nicassio.it/daniele/AsciiArtGenetic/>

Ejercicio 2

*Se busca implementar un compresor de imágenes un tanto peculiar. Deberemos implementar un motor de Algoritmos Genéticos que pueda recibir una imagen, y lograr la mejor aproximación a ella a través de **triángulos** sobre un canvas blanco.*

Nuestros únicos parámetros (no confundir con hiperparámetros) entonces serán la imagen a procesar, y T – la cantidad de triángulos que queremos utilizar para aproximar esa imagen. Los triángulos deberán ser de un color uniforme pero pueden ser traslúcidos (RGBA, HSLA, ...).



Input

- Imagen
- Cantidad de triángulos
- Hiperparámetros de la implementación de Algoritmos Genéticos

Output

- Imagen generada
- Enumeración de triángulos (posición, color, ...)
- Métricas para análisis para defender su implementación (fitness, error, generaciones, etc...)

Implementar y resolver

- Implementar los métodos de selección vistos en clase
- Implementar ambos criterios de selección para crear nuevas generaciones
- Decidir de que manera(s) terminará la ejecución
- Justificar la estructura y la función de aptitud
- Decidir qué método(s) de cruce y mutación utilizarían en diferentes circunstancias y por qué
- (!) Es posible utilizar librerías externas para el manejo de imágenes, pero no para la implementación de Algoritmos Genéticos

Entregable (digital)

- Código fuente
- Presentación
- Un archivo README explicando cómo ejecutar el programa

Opcionales

- La cantidad de triángulos parámetros es la cota máxima, y adicionalmente se recibe un error mínimo para considerar la imagen una buena réplica.
- Otros polígonos en vez de triángulos, u óvalos (x, y, r_x, r_y, θ).

EJERCICIO 1:

Utilizando un algoritmo típico del tipo evolutivo:

```
Target      <-  Obtener_imagen()
Población   <-  Generar_población()

For generación in 1..generaciones:
  Scores      <-  Calcular_score(Población)
  NuevaPoblación <- [MejoresDePoblación]

  While NuevaPoblacion < Población:
    P1, P2     <-  ObtenerPadres(Población)
    Hijo       <-  Cruzar(P1, P2)
    NuevaPoblacion.append(Hijo)

Return Mejor(NuevaPoblación)
```

Se debería prestar atención a las funciones **Calcular_score** y **Generar_población**. Ya que estas son las únicas que son específicas del problema a resolver.

Generar_Poblacion: Genera una estructura de datos que tenga como cromosoma a una cadena de caracteres y las dimensiones de la imagen a crear.

Calcular_score: Debería dar una idea de similitud entre la imagen y los elementos del cromosoma. Para esto se puede decodificar el cromosoma pasando un mapeo de caracteres a su correspondiente valor de intensidad o matriz asociada. Entonces agrupando las submatrices de cada string se puede recrear una matriz comparable a la de la imagen objetivo y calcular una métrica como la norma 1 o 2 entre ambas.

0	255	255	255
0	255	255	255
0	255	255	255
0	0	0	255

Figura 1 - Sub matriz que representa la letra "L"

0	255	255	255	255	255	0	255
0	255	255	255	255	255	0	255
0	255	255	255	255	255	0	255
0	0	0	255	255	255	0	255



Figura 2 - Representación en ASCII (LI) de la figura