Documentación de Computación Bioinspirada

Entrega de Algoritmos Genéticos

Autores: Juan Antonio Silva Luján y Luis Bote Gallardo

Universidad de Extremadura, Escuela Politécnica (Cáceres).

Índice de contenidos:

Introducción a algoritmos genéticos

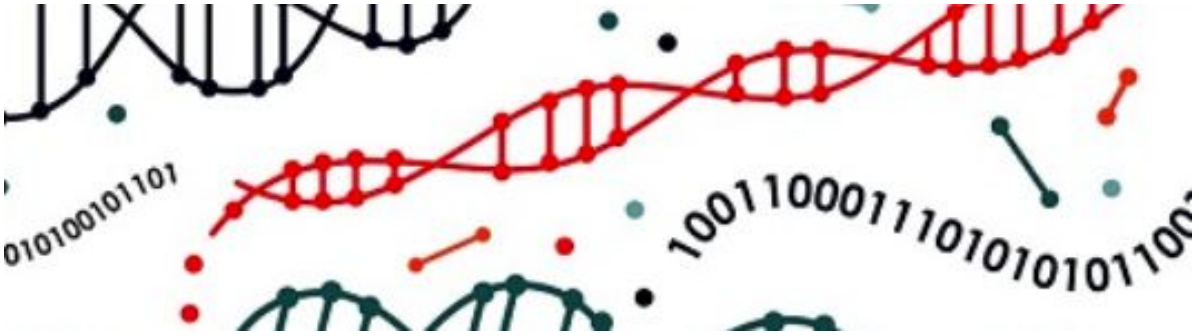
Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización.

Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza de acorde con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas.

Un **algoritmo genético** consiste en una función matemática o un software que toma como entradas a los ejemplares y retorna como salidas cuáles de ellos deben generar descendencia para la nueva generación.

Versiones más complejas de algoritmos genéticos generan un ciclo iterativo que directamente toma a la especie (el total de los ejemplares) y crea una nueva generación que reemplaza a la antigua una cantidad de veces determinada por su propio diseño.

Una de sus características principales es la de ir perfeccionando su propia heurística en el proceso de ejecución, por lo que no requiere largos períodos de entrenamiento especializado por parte del ser humano, principal defecto de otros métodos para solucionar problemas, como los Sistemas Expertos.



Introducción al problema

El problema planteado es el Viajante de Comercio, dónde tenemos que conseguir la ruta más corta para recorrer todas las ciudades dadas por un fichero.

Supondremos que los caminos entre ciudades forman un grafo no dirigido (simétrico) donde los nodos serán las distintas ciudades y los enlaces que conectan estos nodos serán los caminos para llegar de unas ciudades a otras, por tanto, el coste o pesos de estos enlaces será la distancia que existente.

Habrá un mínimo de 10 ciudades con valores relacionados con las distancias entre 1 y 100, además, la ejecución del algoritmo se llevará a cabo durante mínimo 1000 ciclos o generaciones.

Se debe considerar una población de entre 100 y 1000 individuos (fijos o dependiente de la cantidad de ciudades) con los que hay que inicializar el POOL con POB de individuos generados aleatoriamente. Los caminos deben ser ordenador de mejor a peor actualizando así el mejor camino hasta el momento.

Planteamiento del problema

Para ello necesitaremos importar el fichero, crear nuestro POOL con el mismo número de individuos que ciudades que tendremos ordenados de mejor a peor, iremos mutando de diferente forma dichos individuos dependiendo de su posición, e iremos actualizando nuestro POOL con los mejores, tantas veces como queramos, hasta obtener la solución deseada.

Recorreremos el POOL haciendo una reproducción entre CAM(i) y CAM(j) donde el valor i es el índice que se recorrerá y j es un valor totalmente aleatorio contenido entre 0 POB-1.

Dados estos detalles:

- Si i < j entonces CAM(i) es el mejor.
- Si j = j entonces se considerarán como el mismo.
- Si i > j entonces se considerará CAM(i) como el peor.

Implementación del algoritmo en Python

Importación de librerías necesarias:

Importamos la librería numpy para hacer uso de todas las funciones internas así como de la random para las generaciones de números aleatorios.

```
import numpy
from random import randrange
```

Función matriz distancias:

Se mostrará por pantalla los resultados correspondientes de la lectura del fichero **`ciudadesEjemplo.txt`** con los datos relativos a las ciudades que se llevarán a cabo durante la ejecución.
El fichero será procesado y se cargarán sus datos que serán separados en un array.

```
def matrizdistancias():
  # Mostramos por pantalla lo que leemos desde el fichero
  print('>>> Cargamos las matriz del txt')
  filename ='ciudadesEjemplo.txt'

  f = open('ciudadesEjemplo.txt')
  # Primera lectura para obviar la cabecera
  ciudades = f.readline()
  print(ciudades)

  # Cargamos los datos
  data = f.read().strip()

  # Se cierra el fichero
  f.close()

  #Separamos los datos en un array
  matriz = [[int(num) for num in line.strip().split(',')] for line in
data.split('\n')]
  for i in matriz:
    print(i)
  return (matriz,ciudades)
```

Creación de población:

Creamos las variables con las ciudades, sus distancias, y el número de ciudades dadas por el fichero.

```
matriz,ciudades = matrizdistancias()
soluciones = []
npob = len(matriz)
```

Función de selección:

Crea el POOL de forma aleatoria con igual tamaño al número de ciudades del fichero, cogemos del POOL un aleatorio, dependiendo del resultado seleccionaremos uno u otro, teniendo en cuenta un 50% de aleatoriedad.

```
def sel(matriz):
    npob = len(matriz)
    k = randrange(npob)
    if (matriz[c] < matriz[k]):
        return(c,k)
    else:
        opcion = randrange(2)
        if opcion == 0:
            return(k,c)
        elif opcion == 1:
            return(c,k)
```

Función de mutación:

La función de mutación se encarga de intercambiar las ciudades del valor a mutar

```
def mut(c):
  x = randrange(npob-1)
  y = randrange(npob-1)
  if (x != y):
    b = x
    x = y
    y = b
  return (b)
```

Función de rep:

La reproducción es el torneo, se crean distintas opciones a partir del POOL y nos quedaremos con las 3 mejores, en nuestro caso las opciones serán obtenidas a partir de copy, flip copy, copy flip, flip flip, y flip reverse.

```
def rep(c,k):

    x = randrange(npob-1)
    h1 = c[:]
    h2 = c[:X] + C[X:].reverse()
    h3 = c[:X].reverse() + C[X:]
    h4 = c[:X].reverse() + C[X:].reverse()
    h5 = k[:X].reverse() + k[X:].reverse()
    h1.sort(key = distancia)
    h2.sort(key = distancia)
    h3.sort(key = distancia)
    h4.sort(key = distancia)
    h5.sort(key = distancia)
    resultado = [h1,h2,h3,h4,h5]
    resultado.sort(key = distancia) # arreglar

    return (resultado[0],resultado[1],resultado[2])
```

Ejecución inicial:

Ejecutamos el algoritmo 50 veces

```
k=0
pob = matriz
for i in range(1,50):
    pob.sort()
    seleccionado = sel(pob)
    c = seleccionado[0]
    k = seleccionado[1]
    rep(c,k) # a que lo igualo
    mut(c)
    i = i+1
```

Fichero de carga de datos:

Durante el inicio de la ejecución se recurre a la carga, lectura, procesado y cierre de este archivo de texto en el que residen unos datos de ejemplo para llevar a cabo la ejecución del algoritmo. En este caso se presentan las ciudades que aparecen a continuación y los valores asociados a ellas.

```
ciudades =
["Badajoz","Caceres","Merida","Plasencia","Malpartida-de-Caceres","Gevora"]

0,93,66,170,94,7

93,0,75,80,11,85

66,75,0,150,74,65

170,80,150,0,87,163

94,11,74,87,0,88

7,85,65,163,88,0
```

Problemas encontrados

La mayoría de los problemas han sido a la hora de transformar la idea del algoritmo a python, ya que aún teniendo una idea de cómo implementarlo a la hora de transformarlo en python teníamos muchos errores, lo que nos ha retrasado mucho.

No hemos sido capaces de implementar una opción para salir del algoritmo si se queda atascado.

Conclusión sobre la práctica

Ambos miembros del grupo hemos concluído en que como primera toma de contacto, este primera actividad entregable ha resultado ser bastante difícil, no obstante, resulta muy interesante practicar con este tipo de algoritmos puesto que representan una solución muy factible a la hora de resolver problemas como la propuesta en el desarrollo de esta práctica.

La aplicación de problemas de inteligencia artificial aplicadas en algoritmos genéticos debidos a su naturaleza evolutiva biológica resulta muy novedosa para nosotros ya que nunca habíamos trabajado con algoritmos de este tipo.