Facultad de Ingeniería Arturo Narro Siller

UAT

VERDAD, BELLEZA, PROBIDAD

DISEÑO ELECTRONICO BASADO EN SISTEMAS EMBEBIDOS

Profesor: Dr. Alejandro H. García Ruiz

Explicación

Algoritmo Genético

ALUMNO:

Zubiri Valdez Hedson Leonardo

MATRÍCULA:

2183330170

Semestre 8° *Grupo: "G"*

27 / 02 / 2022

Algoritmo genético para binarios

```
def calc_FO(indv):
    return sum(indv)
```

Función objetivo. – Retorna la cantidad de 1's en una lista. En este caso lo hace mediante el retorno de la suma de todos los elementos del individuo (indv).

```
#m = numero de genes
tot_genes = 10
#n  = numero de vectores
tot_individuos = 100  #numero de individuos
```

Se establece que habrá 100 individuos los cuales contienen genes en este caso serán 10 elementos (0´s y 1´s) en una lista por cada uno.

```
#Poblacion Inicial
import random as rnd
poblacion = []
```

Se importa librería para generar números aleatorios.

Población se almacenará en una lista.

```
for i in range(tot_individuos):
```

For de 100 iteraciones

```
   vector = [rnd.randint(0,1)  for i in range(tot_genes)]
```

Se genera vector con comprehension lists de 0´s y 1´s representando los 10 genes de un individuo.

```
   poblacion.append([vector, calc_FO(vector)])
```

Se agrega el nuevo individuo generado a la población con el formato [[10 genes], Cantidad de 1´s en genes]

```
#print("Poblacion Inicial: ")
#for indv in poblacion:
#    print(indv)
```

En caso de querer imprimir cada individuo generado y almacenado en población, ejemplo: [[**1**, 0, **1**, 0, **1**, 0, 0, **1**, 0, 0], **4**] ….

```
padres = []
tot_padres = 50
```

Una lista donde se almacenarán los padres así como la cantidad total de estos.

```
poblacion.sort(key= lambda x:x[1], reverse=True)
#sorted(poblacion, key= lambda…)
#Sorted no afecta original
```

.sort( ) afecta a la lista original y se utiliza para ordenar mediante lambda, x:x[1] significa:

El parámetro x, es el elemento por el cual se ordenará = x[1] representa que será por la FO de los genes

[0] [1]
x [[1, 0, 1, 0, 1, 0, 0, 1, 0, 0], **4**]

```
#Me quedo con los MejoresPadres
poblacion = poblacion[0:tot_padres]
print("tot poblacion de mejores: ",len(poblacion))


for i in range(tot_padres):
    indexPadre1 = rnd.randint(0, tot_padres-1)
    indexPadre2 = rnd.randint(0, tot_padres-1)
    while(indexPadre1==indexPadre2):
        indexPadre2 = rnd.randint(0,tot_padres -
1)

    tempPadre1 = poblacion[indexPadre1][0]
    tempPadre2 = poblacion[indexPadre2][0]

    if calc_FO(tempPadre1) >= calc_FO(tempPadre2):
        padres.append(tempPadre1.copy())
    else:
        padres.append(tempPadre2.copy())

#print("Padres para cruza: ")
#for index, padre in enumerate(padres):
#    print(index,".-", padre)


hijos = []
for i in range(0,tot_padres, 2):
    tempPadre1 = padres[i]
    tempPadre2 = padres[i+1]

    #Generar un aleatorio
    puntoCruza = rnd.randint(0, tot_genes-1)

    # Generar un aleatorio
    puntoCruza += 1  # +1 -> puntoCruza incluyente

 hijo1 = tempPadre1[:puntoCruza]+tempPadre2[puntoCruza:]
 hijo2 = tempPadre2[:puntoCruza]+tempPadre1[puntoCruza:]


    hijos.append([hijo1, 0])
    hijos.append([hijo2, 0])
```

Al tener ordenada la población de manera descendente (reverse = true), los 50 primeros son los que mejor puntaje de FO tienen. Entonces mi población ahora es desde la posición 0 a tot_padres(50).

Se generan dos índices que representan individuos que serán padres, valores desde 0 a 49, si se generan los mismos índices el segundo seguirá generándose hasta que sea distinto.

Se recupera el individuo de la población, pero solo la parte de los genes [0], sin FO([1]).

Se calcula cual padre tiene mayor numero de 1´s mediante la FO, y ese se agrega a la lista de padres

En caso de querer imprimir los padres, se utiliza enumerate para conseguir un index además del contenido de la lista con genes.

Los hijos se agregarán a una lista.

El for itera de 2 en 2 hasta 50, para elegir así pares de padres (pareja)

Se genera el punto desde el cual se establecen las porciones de los genes a mezclarse para los hijos.

El punto se aumenta para poder hacer un slice correctamente.

Hijo1=1erPartePadre1 + 2daPartePadre2

Hijo2=1erPartePadre2 + 2daPartePadre1

Se agregan a la lista de hijos, con el formato de [genes], FO. En este caso se deja en 0

```
#Mutacion
probMuta = 0.8
```

Existe una probabilidad del 80% que cada gen del hijo pueda cambiar aleatoriamente.

```
for indexHijo in range(len(hijos)):

    hijo = hijos[indexHijo][0]

    for indexGen in range(len(hijo)):

        r = rnd.random() # 0 - 1

        if r >= probMuta:

            #se efectua la mutacion
            val = 1 if rnd.random() >= 0.5 else 0
            #print(val)

            hijo[indexGen] = val



    hijos[indexHijo][1] = calc_FO(hijo)

##poblacion completa
#hijos

print(len(hijos))
```

Se recorren los index de la cantidad de hijos

Por cada iteración se obtiene los genes del hijo correspondiente [0]

Se recorren index de los genes del hijo

Se obtiene un numero aleatorio entre 0 y 1

Si es mayor o igual a 0.8 quiere decir que mutará

Se genera un numero random, si es >= que 0.5 será 1, si no 0. El resultado se guarda en val.

Se actualiza el gen con el valor generado.

El 0 que por defecto se había puesto se cambia por el verdadero cálculo de la FO, para seguir con el formato:

[[10 genes], Cantidad de 1´s en genes]

[indexHijo] [1]
x [[1, 0, 1, 0, 1, 0, 0, 1, 0, 0], **4**]