Universidad Autónoma del Estado de México

Ingeniería en Computación

Materia: Algoritmos Genéticos
Laboratorio 03

Axel Valenzuela Juárez
Profesor: Dr. Asdrúbal López Chau
21 de Febrero del 2020

1. Introduccion

Los Algoritmos Genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos.

Los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución optima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de los algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia.

El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de estas hibridándolas con los Algoritmos Genéticos.

2. Desarrollo

La implementación del primer algoritmo genético del curso se realizo de manera diferente en este caso, ya que se nos fue presentado el algoritmo de un fragmento de libro el cual debíamos leer y codificar según los pasos del libro.

El algoritmo esta compuesto de 4 funciones una función para generar un padre, otra para obtener la aptitud, una función para mutar a los hijos, y una ultima encargada de mostrar los datos. La primer función llamada *generarpadre*() **se encarga de generar una cadena de caracteres de la misma longitud que el objetivo.**

Para hacer esto primero es necesario que se aplique un while que permite hacer este procedimiento siempre que no se exceda de la longitud del objetivo que en este caso era hola mundo.

A continuación de le restará la longitud del objetivo con la longitud del arreglo genes y se obtendrá el mínimo del resultado de lo anterior con el genset. Ref: 1

```
10
11 import random
12
13 def generar_padre(longitud):#pide un entero
14     genes=[]#arreglo
15     while len(genes) < longitud: #while de tamaño de genes es menor que longitud
16         tamanioMuestral = min(longitud - len(genes),len(geneSet))#len es la longitud, min e
17         genes.extend(random.sample(geneSet,tamanioMuestral))#genera una lista aleatoria de
18         #extend agrega los datos al final de la lista de genes
19         print(genes)
20         return ''.join(genes) #convierte a string los genes y les da un espacio con el '' q
21
```

Figura 1: Metodo generarpadre.

Al final de este método se imprime el gen y se regresa convertido en string con un espacio entre caracteres.

```
22
23 def obtener_aptitud(conjetura):
24     return sum(1 for esperando, real in zip(objetivo,conjetura)#zip junta objetivo con
25                if esperando==real)
```

Figura 2: Metodo obteneraptitud.

El segundo método llamado *obteneraptitud*() **recibe una variable nino la cual contiene una mutación , el objetivo de la función es juntar el objetivo y la conjetura para después sumarlo en un for, esto se regresa a la función mostrar para que los datos se muestren en pantalla. Ref: 2**

```
26
27 def mutar(padre):
28     indice=random.randrange(0,len(padre))#crea datos aleatorios en el rango de 0 al tamañ
29     genesDelNino = list(padre)#crea una lista de los valores de padre
30     nuevoGen,alterno=random.sample(geneSet,2)#les da valores aleatorios de el rango de ge
31     genesDelNino[indice]=alterno if nuevoGen==genesDelNino[
32             indice] else nuevoGen#se mete en la lista genesnino alterno si nuevo gen es ig
33     return ''.join(genesDelNino)#se regresa un string con los datos de genes de nino sepa
34
35 import datetime#para trabajar con fechas
```

Figura 3: Función mutar

La siguiente función llamada mutar() es una de las funciones principales y mas importantes ya que permite hacer variaciones en el gen, esto funciona a partir de crear datos aleatorios y guardarlos en un índice. Después se crea otros valores aleatorios con el rango de los genes para poder cambiar la cadena e intentar mejorar el gen, el resultado se regresa como string y se le da un espacio entre valores. Ref: 3

```
7 def mostrar(conjetura):
8     diferencia=(datetime.datetime.now() - horaInicio).total_seconds()#la fecha actual men
9     aptitud= obtener_aptitud(conjetura)#se manda a llamar la funcion obtener_aptitud y se
0     print("{}\t{}\t{}".format(conjetura,aptitud,diferencia))#se imprimen en el formato en
1     #se le pasa la conjetura,aptitud y diferencia
```

Figura 4: Función mostrar.

La función de mostrar() se encarga de mostrar los datos en pantalla de buena manera, obtiene la diferencia de tiempo entre cada gen, muestra también la conjetura que en si es la forma en que se va acercando el algoritmo a el objetivo y muestra también la aptitud todo eso se muestra en un formato de tres. Ref: 4

```
42
43 random.seed()#se gnera la semilla
44 horaInicio=datetime.datetime.now()#se obtiene hora actual
45 mejorPadre= generar_padre(len(objetivo))#se manda a llamar la funcion generar padre y
46 mejorAptitud=obtener_aptitud(mejorPadre)#se guarda el resultado de la funcion obtener
47 mostrar(mejorPadre)#se muestra el contenido de mejorPadre
48
49 while True:
50     nino=mutar(mejorPadre)#se muta al mejorpadre
51     ninoAptitud = obtener_aptitud(nino)#se manda a obtener aptitud a la mutacion llam
52     if mejorAptitud >= ninoAptitud:#se comparan para saber si se debe continuar
53         continue
54     mostrar(nino)#se manda a llamar la funcion mostrar para que se muestre los valore
55     if ninoAptitud >= len(mejorPadre):#si ninoAptitud es mayor o igual que mejorPadre
56         break
57     mejorAptitud = ninoAptitud#se pone que ninoAptitud es igual a la mejor actitud
58     mejorPadre=nino #que el nino es igual a el mejor padre y se sigue con el programa
```

Figura 5: Main.

Las partes del código que no están adentro de una función son las que normalmente estarían adentro de un main, se encargan de pasar los valores a las funciones y así como se hacen las comparaciones necesarias para saber si el programa debe seguir o terminar e ir agregando la mejor aptitud a un mejor padre. Ref: 5

```
In [1]: runfile('C:/Users/Valenzuela/Documents/
AlgoritmosGeneticos/Lab03.py', wdir='C:/Users/Valenzuela/
Documents/AlgoritmosGeneticos')
['f', 'a', 's', 'o', 'z', 'w', 'y', 'q', 't', 'n', 'r', 'k']
fasozwyqtnrk    0       0.0
fhsozwyqtnrk    1       0.015624
fhsozwmqtnrk    2       0.015624
fhsoz mqtnrk    3       0.015624
fhsoz mqtnr!    4       0.015624
ihsoz mqtnr!    5       0.015624
ihsoz mqnnr!    6       0.015624
ihsoz munnr!    7       0.015624
ihsoa munnr!    8       0.015624
ihooa munnr!    9       0.015624
ihooa mundr!    10      0.015624
ihola mundr!    11      0.015624
ihola mundo!    12      0.015624

In [2]:
```

Figura 6: Resultados.

En los resultados se podía observar lo genial que funciono el algoritmo al lograr encontrar la clave de una manera muy rápida. Ref: 6

3. Conclusión

Esta practica me mostro el poder de los algoritmos genéticos, tener un algoritmo que puede evolucionar hasta el punto en que dé solución a un problema en especifico hace que tenga miles de aplicaciones, tanto para encontrar claves como en esta práctica, como para mejorar procesos y ayudar a algoritmos a tener una mejor eficiencia.

Esta práctica despertó aún más mi interés en los AG, Y a la vez aprendí que Python realmente es muy poderoso, ya que implementar este algoritmo en otro lenguaje como java se volvió bastante complicado en mi caso ya que me encontré con funciones que no existen en java y obligan a hacer mucho más trabajo de código que en Python se resume en una línea.

Al terminar de comprender todo el código logre ver que realmente no hay mucha complejidad en la manera en que trabaja este algoritmo, simplemente se busca mejorar siempre a los hijos y si es necesario se mutan para tener probabilidades de mejorar aun mas y se van escogiendo a los mejores hasta obtener el objetivo.

4. Referencias

(2019). Obtenido de sc.ehu.es: www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/temageneticos.pdf

(2020). Obtenido de w3schools: www.w3schools.com/python/pythondatetime.asp

(2020). Obtenido de Programiz: www.programiz.com/python-programming/methods/built-in/list

(2019). Obtenido de Oracle: docs.oracle.com/javase/8/docs/api/java/util/Random.html

(2017). Obtenido de picodotdev: picodotdev.github.io/blog-bitix/2017/10/obtener-el-minimo-o-maximo-de-dos-una-lista-o-stream-de-valores-en-java/