Algoritmo genético aplicado a función de 10 variables

Ciro Fabian Bermudez Marquez
INAOE
Mexico, Puebla
cirofabian.bermudez@gmail.com

***Resumen*—En este trabajo se pone a prueba el algoritmo genético (AG) para un problema de minimización de una función de 10 variables implementado en python.**

***Index Terms*—AG, python, heuristica.**

I. Introducción

Los métodos de búsqueda heurística consisten en añadir información, basándose en el espacio estudiado hasta ese momento para hacer una búsqueda inteligente. Para problemas no lineales y multimodales se justifica usar heurísticas.

Las heurísticas se deben usar si el problema tiene tres o más variables y en este trabajo se estudiará el algoritmo genético (AG) aplicado a una función de 10 variables.

$$f_1(x) = 0.1 \sum_{i=1}^{10} (x_i - 2)(x_i - 5) + \sin(1.5\pi x_i) \qquad (1)$$

teniendo en cuenta las siguientes características para el algoritmo $p_c = 0.7$, $p_m = 0.1$, $\mu = 200$, $g = 50, 100, 150$, codificación de 10 bits, espacio de búsqueda $x_i = [0, 7]$ para todas las $i = \{1, 2, \ldots, 10\}$.

II. El algoritmo genético

Para entender el algoritmo genético es necesario conocer la terminología que se utiliza.

Individuo: es un vector que codifica las variables del problema.

Población: una matriz, un conjunto de vectores.

Las variables para el AG se codifican en cadenas binarias, de 0 o 1. El AG codifica problemas combinarlos, no continuos.

Para cambia una cadena binaria a una variable real, se puede mapear linealmente:

$$\frac{x - \min_x}{\max_x - \min_x} = \frac{b}{2^p - 1} \qquad (2)$$

donde la precisión es igual a

$$\text{precisión} = \frac{\max_x - \min_x}{2^p} \qquad (3)$$

y p es el número de bits.

Para la representación binaria se utiliza la codificación Grey, ya que esta solo cambia un bit entre cada valor.

A grandes rasgos la filosofía del algoritmo genético se puede resumir en los siguientes pasos:

- Utiliza un conjunto de soluciones (se le llama población).
- Combina soluciones para generar otras.
- La solución que sobrevive es la mejor (la más apta).
- Usa los operadores de selección, cruza, mutación y elitismo.
- La población guarda la *inteligencia* del algoritmo.

El algoritmo requiere las siguientes variables para su funcionamiento:

- Tamaño de población μ
- Numero de generaciones g
- Probabilidad de cruza p_c
- Probabilidad de mutación p_m
- La función a optimizar.

El AG realiza las siguientes operaciones:

1. Se inicializa aleatoriamente la población
2. Se evaluá la población
3. Para un número de generaciones:
   a) Se seleccionan dos individuos
   b) Se cruzan
   c) Su mutan y se evalúan los hijos
   d) Se aplican elitismo
   e) La población de hijos sustituye a la de padres
4. Se reporta el mejor individuo

II-A. Detalles de pasos de AG

La selección, la cruza, la mutación y el elitismo son operadores genéticos.

La selección de los individuos se realiza por medio de un **torneo binario**.

i1 = Se escogen dos individuos y gana el mejor

i2 = Se escogen dos individuos y gana el mejor

Al inicio de cada iteración se revuelven aleatoriamente la población (se barajean los índices). Y se van tomando de dos en dos.

La **cruza de dos puntos** consisten en intercambiar partes de las cadenas binarias entre padres e hijos y se usa una probabilidad de cruza en [0,1].

En la **mutación** se escoge aleatoriamente una posición de la cadena y se invierte el bit. Se aplica con un *probabilidad de mutación* que toma valores en [0,1].

Algo importante a resaltar es que si la probabilidad de cruza y mutación son iguales a 1, el algoritmo está haciendo una

búsqueda aleatoria. Es recomendado que la probabilidad de mutación este en un valor menor que 0.2.

El **elitismo** consiste en guardar la mejor solución para garantizar convergencia, el mejor individuo se mantiene en la población.

El precio de usar una heurística es que la función del problema se tiene que ejecutar el número de generaciones multiplicada por el número de individuos.

III. Resultados

Lo primero que hay que hacer es cambiar la función objetivo del archivo **evalua.py** de la siguiente manera:

```
import math

def Evalua( _n, vpar ) :
  suma = 0
  for i in range(_n):
    suma = suma +  (vpar[i]-2)*(vpar[i]-5) + math.
    sin( 1.5*math.pi*vpar[i])
    v = 0.1*suma
  return v
```

Código 1. Generar número aleatorio flotante.

Y haciendo las modificaciones al archivo **corre.py** con las especificaciones del problema se obtuvo lo siguiente:

Tabla I
PUNTOS DONDE HAY UN MÁXIMO O MÍNIMO.

# Generaciones	Mínimo
50	-3.21680104689738
100	-3.224391005143813
150	-3.224504148622877

Sabemos que el mínimo esta en con $x = 3.652887442162$ $f_1(x) =$ -3.224518019 y con 150 generaciones ya nos aproximamos bastante al resultado real.

IV. Conclusiones

El algoritmo encuentra un resultado aceptado cuando utilizamos 150 generaciones, y tiene la ventaja de no necesitar las derivadas de la función objetivo en contraposición con el método de Newton, sin embargo este tiene la desventaja de tener que elegir correctamente los parámetros del algoritmo y que dependiendo de ellos podemos tener mejores o peores resultados, con un poco de práctica y con experiencia utilizando este tipo de algoritmos su utilidad se ve inmediatamente al notar que para resolver un problema de 10 variables encuentra una solución relativamente rápido.

Referencias

[1] Dr. Luis Gerardo de la Fraga. “Apuntes de clase” .