Instituto Politécnico Nacional
Escuela Superior de Cómputo
Algoritmos Bioinspirados
Practica 2 – Algoritmos Genéticos
Tinoco Videgaray Sergio Ernesto
Jorge Luis Rosas Trigueros
31/10/2022

Marco teórico:

En el área de la inteligencia artificial existe un campo que se dedica a encontrar la solución óptima a diversos problemas por medio de algoritmos que se basan en el comportamiento de la naturaleza a lo largo del tiempo, algunos de estos algoritmos se les conoce como algoritmos evolutivos o también llamados “algoritmos genéticos” que como su nombre lo indica, replican el comportamiento de los genes o genomas, los cuales proporcionan información a base de un código el cual se encuentra almacenado en los conocidos “cromosomas”.

Estos algoritmos se caracterizan por su comportamiento iterativo y exploración de soluciones en base a alteraciones en los genomas. Ya que estos algoritmos siguen una serie de pasos que se ilustran en el diagrama 1.

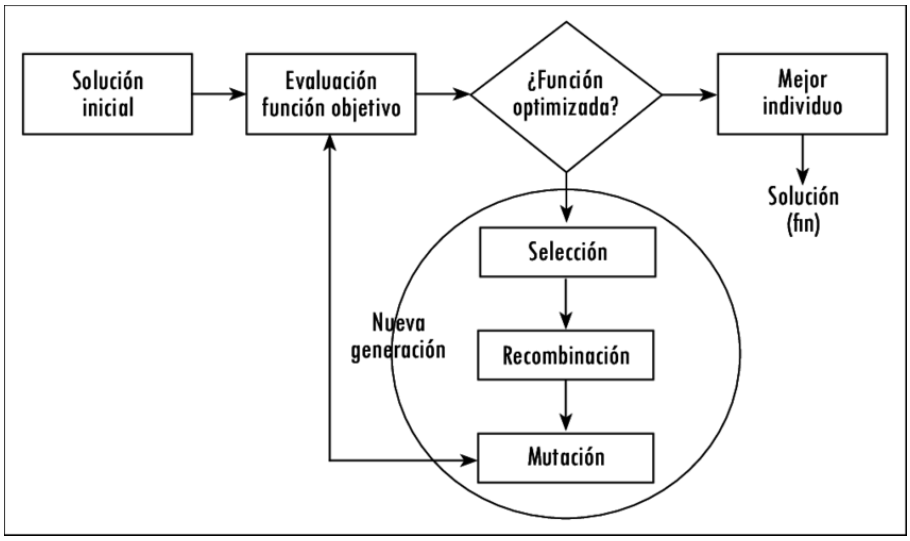

Diagrama 1 Algoritmo Genético en cada una de sus etapas.

Cada iteración de este algoritmo se le conoce como generación, en cada generación se “generan” nuevos cromosomas a partir de los que ya están realizando alguna operación de recombinación como un desplazamiento, un corte y empalme, un intercambio en uno o varios puntos, etc. Y posteriormente a esta recombinación se realiza una mutación, que no es más que cambiar el valor de uno de los cromosomas con el fin de explorar nuevas soluciones.

Estos algoritmos son muy útiles cuando se busca encontrar alguna solución óptima a cierta función, ya que su capacidad para “explorar” nuevas soluciones da lugar a que el algoritmo pueda encontrar aquella solución óptima para la función sin el riesgo de quedar atascada en algún punto del conjunto, lo que los vuelve muy versátiles en la mayoría de los casos.

Función de Rastrigin.

La función de Rastrigin *(ecuación 1)* fue propuesta en 1974 por Rastrigin, es utilizada comúnmente para problemas de rendimiento, es decir, esta función permite realizar pruebas de optimización. Esto debido a su comportamiento multimodal, lo que quiere decir que posee una gran cantidad de mínimos locales en su dominio *(ver figura 1.1 y figura 1.2)*.

$$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right]$$

Ecuación 1 función de Rastrigin en su forma general

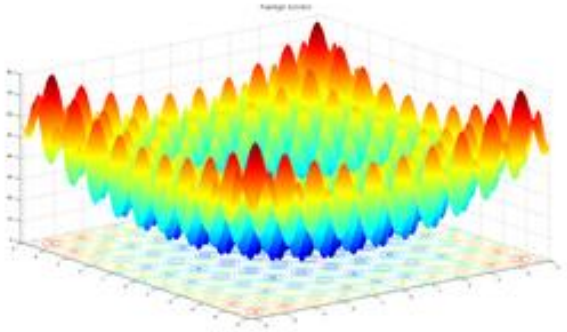

Figura 1.1 Grafico de la función de Rastrigin en el plano X, Y, Z.

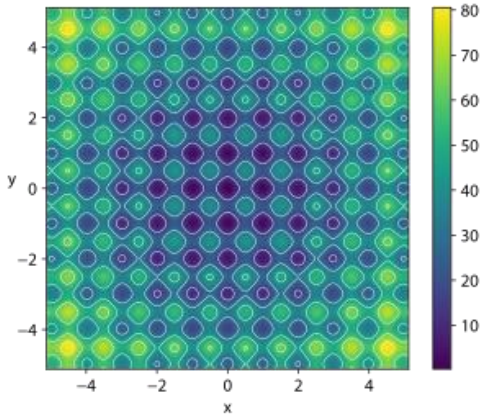

Figura 1.2 Grafico de la función de Rastrigin en el plano X, Y.

En el desarrollo de esta práctica se va a implementar un algoritmo genético que optimice la función de Rastrigin para 2 variables x1, x2.

Material y equipo:

-Equipo: Computadora con conexión a internet.

-IDE: Google Colab.

-Lenguaje de programación: Python 3.6

Desarrollo de la practica:

Optimización de la función de Rastrigin.

Implementación en Python.

Para la implementación de este algoritmo se utilizó un algoritmo genético que va generando sus propios cromosomas para descubrir algún espacio en el conjunto que localice algún punto mínimo.

Para su implementación en Python se utilizaron algunas bibliotecas como openturns, numpy, random, matplotlib, functools e ipywidgets.

Primeramente, se instala el paquete de openturns en Colab.

```
!pip install openturns
```

Se define la funcion de Rastrigin con base en la ecuación 1.

```
def f(X):
    A = 10.0
    delta = [x**2 - A * np.cos(2 * np.pi * x) for x in X]
    y = A + sum(delta)
    return [y]
```

Se crea una funcion para generar un botón que permita pasar a la siguiente generación.

```
def create_button():
  button = widgets.Button(
    description='Next Generation',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Next Generation',
    icon='check' # (FontAwesome names without the `fa-` prefix)
  )
  return button
```

Se define el tamaño de los cromosomas para el genotipo, para este caso se define un tamaño de 4, y posteriormente se crea una variable para la conversión de los cromosomas.

```
L_chromosome=4
N_chains=2**L_chromosome
```

Se definen los limites inferior y superior, así como el punto del crossover.

```
a=-5
b=5
crossover_point=int(L_chromosome/2)
```

Posteriormente se define una función que genera una lista de valores binarios de forma aleatoria para cada cromosoma.

```
def random_chromosome():
    chromosome=[]
    for i in range(0,L_chromosome):
        if random.random()<0.1:
            chromosome.append(0)
        else:
            chromosome.append(1)

    return chromosome
```

Se define el número total de cromosomas y la probabilidad de que estos muten.

```
N_chromosomes=10
prob_m=0.15
```

Se define una lista para almacenar los 10 cromosomas y de igual forma se define un vector para los valores de ajuste.

```
F0=[]
fitness_values=[]
```

Posteriormente se agregan a la lista los cromosomas generados aleatoriamente.

De igual forma se agregan 0’s al vector de ajuste.

```
for i in range(0,N_chromosomes):
    F0.append(random_chromosome())
    fitness_values.append(0)
```

Se define la función para decodificar la cadena del cromosoma.

```
def decode_chromosome(chromosome):
    global L_chromosome,N_chains,a,b
    value=0
    for p in range(L_chromosome):
        value+=(2**p)*chromosome[-1-p]

    return ((b-a)*float(value)/(N_chains-1))+a
```

Se define una función para evaluar cada cromosoma y agregarlo al vector de ajuste.

```
def evaluate_chromosomes():
    global F0

    for p in range(N_chromosomes):
        v1=decode_chromosome(F0[p])
        fitness_values[p]=f([v1])
```

De igual forma se define una función para comparar los cromosomas.

```
def compare_chromosomes(chromosome1,chromosome2):
    vc1=decode_chromosome(chromosome1)
    vc2=decode_chromosome(chromosome2)

    fvc1=f([vc1])
    fvc2=f([vc2])

    if fvc1 > fvc2:
        return 1
    elif fvc1 == fvc2:
        return 0
    else: #fvg1<fvg2
        return -1
```

Se define el tamaño de la rueda.

```
Lwheel=N_chromosomes*10
```

Se crea la función para generar la ruleta usando el tamaño de la rueda definido previamente.

```
def create_wheel():
    global F0,fitness_values

    maxv=max(fitness_values)
    acc=0
    for p in range(N_chromosomes):
       acc+=maxv-fitness_values[p][0]
    fraction=[]
    for p in range(N_chromosomes):
        fraction.append( float(maxv-fitness_values[p][0])/acc)
        if fraction[-1]<=1.0/Lwheel:
            fraction[-1]=1.0/Lwheel
##    print fraction
    fraction[0]-=(sum(fraction)-1.0)/2
    fraction[1]-=(sum(fraction)-1.0)/2
##    print fraction

    wheel=[]

    pc=0

    for f in fraction:
        Np=int(f*Lwheel)
        for i in range(Np):
            wheel.append(pc)
        pc+=1

    return wheel
```

Se crea una copia de la lista de cromosmas.

```
F1=F0[:]    #Copia de F0
```

Se define una función para generar la siguiente generación.

```
def nextgeneration(b):
```

Se muestra el botón para la siguiente generación.

```
    display.clear_output(wait=True)
    display.display(button)
```

Se ordena la lista de cromosomas haciendo una comparación de cromosomas.

```
    F0.sort(key=cmp_to_key(compare_chromosomes) )
```

Imprime la mejor solución para el algoritmo que fue colocada en la primera posición de la lista.

```
print( "Best solution so far:")
print( "f(",decode_chromosome(F0[0]),")= ", f([decode_chromosome(F0[0])]) )
```

Se asignan los mejores cromosomas a la copia de la lista original.

```
   F1[0]=F0[0]
    F1[1]=F0[1]
```

Por medio de la ruleta previamente creada, se van a seleccionar aleatoriamente los 2 padres.

Y posteriormente se generan los descendientes haciendo una mezcla de los cromosomas correspondientes por medio del punto de crossover.

```
    for i in range(0,int((N_chromosomes-2)/2)):
        roulette=create_wheel()
        #Two parents are selected
        p1=random.choice(roulette)
        p2=random.choice(roulette)
        #Two descendants are generated
        o1=F0[p1][0:crossover_point]
        o1.extend(F0[p2][crossover_point:L_chromosome])
        o2=F0[p2][0:crossover_point]
        o2.extend(F0[p1][crossover_point:L_chromosome])
```

Se realiza la mutación de los nuevos descendientes por medio de la probabilidad.

En caso de realizar la mutación se aplica el operador XOR a uno de los bits.

```
if random.random() < prob_m:
     o1[int(round(random.random()*(L_chromosome-1)))]^=1   #XOR 1
if random.random() < prob_m:
     o2[int(round(random.random()*(L_chromosome-1)))]^=1   #XOR 1
```

Se añaden los nuevos descendientes a la copia de los cromosomas.

```
        F1[2+2*i]=o1
        F1[3+2*i]=o2
```

Finalmente se grafican las posibles soluciones con el nuevo arreglo de cromosomas y se remplaza

```
    graph_population(F1)
```

Se remplaza al arreglo original con la nueva generación.

```
    F0[:]=F1[:]
```

Se define la función para graficar la función de Rastrigin

```

def graph_f():
    xini=-5
    xfin=5

    rastrigin = ot.PythonFunction(2, 1, f)

    rastrigin = ot.MemoizeFunction(rastrigin)

    lowerbound = [xini] * 2
    upperbound = [xfin] * 2
    bounds = ot.Interval(lowerbound, upperbound)

    graph = rastrigin.draw(lowerbound, upperbound, [100] * 2)
    graph.setTitle("Rastrigin function")
    view = viewer.View(graph, legend_kw={"bbox_to_anchor": (1, 1), "loc": "upper
left"})
    view.getFigure().tight_layout()
```

Se define una función para graficar las posibles soluciones.

```
def graph_population(F):
    x=list(map(decode_chromosome,F))
    graph_f()
    plt.plot(x,y_population,'go')
```

Se generan los botones correspondientes.

```
button=create_button()
button.on_click(nextgeneration)
display.display(button)
```

Se decodifican los 10 cromosomas y se almacenan en una variable x.

```
x=list(map(decode_chromosome,F0))
```

Se genera una matriz de 0's de 10x10.

```
y_population=np.zeros([N_chromosomes,N_chromosomes
```

Se grafica la función de Rastrigin

```
graph_f()
```

Se plotea la primer posible solución y se evalúan los cromosomas generados.

```
plt.plot(x,y_population,'go')
F0.sort(  key=cmp_to_key(compare_chromosomes))
evaluate_chromosomes()
```

Resultados

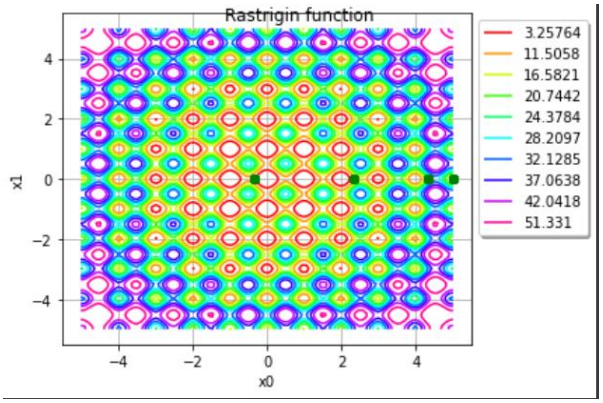

Imagen 1. función de Rastrigin con la primera generación.

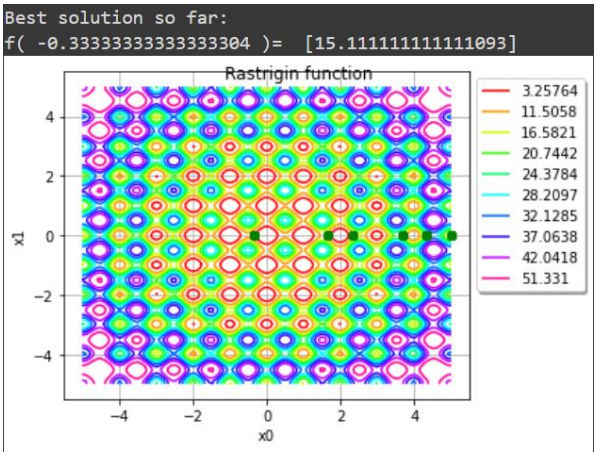

Imagen 2. función de Rastrigin con la segunda generación.

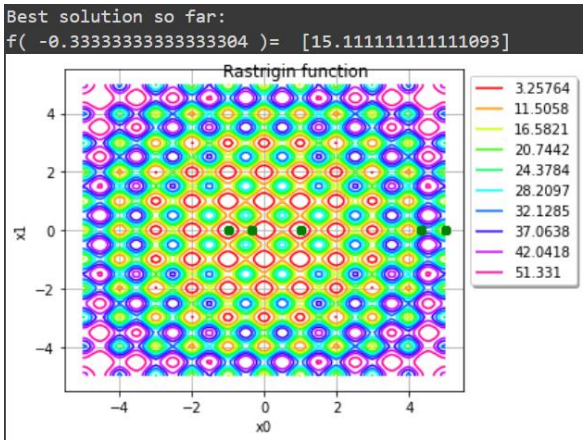

Imagen 3. función de Rastrigin con la tercera generación.

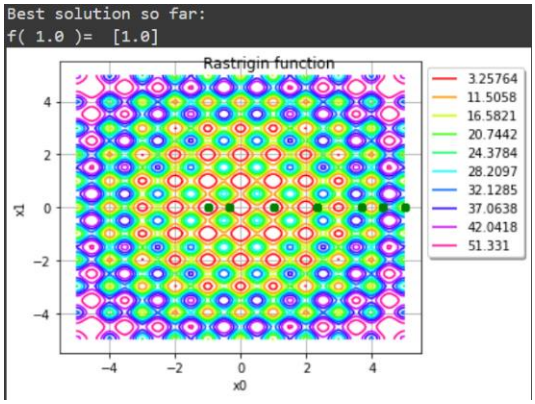

Imagen 4. función de Rastrigin con la cuarta generación.

Conclusiones:

Los algoritmos genéticos nos permiten encontrar soluciones en un espacio n-dimensional realizando una k cantidad de iteraciones que se van dando de forma aleatoria y nos ayudan a buscar mas soluciones de las que podemos identificar. Yo personalmente no conocía estos tipos de algoritmos y se me hizo interesante la manera en como operan y encuentran soluciones basadas en una gran cantidad de parámetros como la probabilidad de mutación, la probabilidad de que un cromosoma sea elegido en la ruleta basado en el valor de ajuste, etc. Y de igual forma me llama la atención de que este método puede encontrar puntos mínimos en una función sin necesidad de aplicar una operación de derivación, ya que como ingenieros nos acostumbramos a aplicar derivadas para buscar puntos críticos en una función y el conocer un nuevo método de optimización para funciones sin derivación hace que este algoritmo se vuelva versátil a pesar de su complejidad computacional.

Como practica personal decidí modificar los valores de la probabilidad de mutación a un 0.1 y la cantidad de cromosomas a 15 y pude observar que los valores tienden a establecerse en un solo punto por lo que se requirió ejecutar más iteraciones del algoritmo para que los puntos pudieran explorar otras soluciones. Por lo que el numero de iteraciones del algoritmo depende de la probabilidad de mutación de los cromosomas.

Bibliografía:

Wikipedia contributors. (2022, 3 febrero). *Rastrigin function*. Wikipedia. https://en.wikipedia.org/wiki/Rastrigin_function

Rastrigin Function. (s. f.). Recuperado 31 de octubre de 2022, de https://www.sfu.ca/%7Essurjano/rastr.html

Optimization of the Rastrigin test function — OpenTURNS 1.20rc1 documentation. (s. f.). Recuperado 31 de octubre de 2022, de https://openturns.github.io/openturns/latest/auto_numerical_methods/optimization/plot_optimization_rastrigin.html