Instituto Politécnico Nacional

Escuela Superior de Cómputo

Evolutionary Computing

Lab Session 6: Particle Swarm Optimization

Alumno: David Arturo Oaxaca Pérez

Grupo: 3CV11

Profesor: Jorge Luis Rosas Trigueros

Fecha de realización: 21/10/2021

Fecha de entrega: 22/10/2021

Marco teórico

La optimización de enjambre de partículas fue propuesta por primera vez en 1995 por Kennedy y Eberhart, este es uno de los algoritmos inspirados por la naturaleza, como puede ser una escuela de peces o una parvada de aves, ya que, como en estos ejemplos, un miembro del grupo puede separarse para buscar comida o refugio y puede informarle al grupo de su descubrimiento para que tengan el mejor resultado en su búsqueda.

Este es un algoritmo relativamente simple para buscar la solución más óptima en un espacio y es diferente a otros algoritmos de optimización en la manera en que solo las funciones objetivo son necesitadas y no son tan dependientes de un gradiente o alguna forma diferencial del objetivo.

Para este algoritmo se considera un enjambre de P partículas, con un vector de posición x y un vector de velocidad y en un tiempo t de iteración para cada partícula que compone el enjambre.

$$X_i^t = (x_{i1}\ x_{i2}\ x_{i3}\ \dots\ x_{in})^T$$

$$V_i^t = (v_{i1}\ v_{i2}\ v_{i3}\ \dots\ v_{in})^T$$

Estos vectores se irán actualizando a través de la dimensión j de acuerdo con la siguiente ecuación:

$$V_{ij}^{t+1} = \omega V_{ij}^t + c_1 r_1^t \left(pbest_{ij} - X_{ij}^t\right) + c_2 r_2^t \left(gbest_j - X_{ij}^t\right)$$

En la ecuación anterior, podemos ver que hay tres contribuciones en el movimiento de una partícula a lo largo de una iteración.

Y $$X_{ij}^{t+1} = X_{ij}^t + V_{ij}^{t+1}$$

Esta segunda ecuación se puede ver como se actualiza la posición de una partícula, donde ω es la inercia constante en esta versión del algoritmo y es un positivo constante y es de importancia como parámetro en la exploración, tanto en la búsqueda global como con la global.

El algoritmo para la implementación de la optimización por enjambre de particulas las siguientes indicaciones:

Inicialización

Para cada particula i en un enjambre con una población de tamaño P:

Inicialización del vector X_i de manera aleatoria

Inicialización del vector V_i de manera aleatoria

Evaluación de la función fitness f(X_i)

Inicialización de gbest con una copia de X_i con el mejor fitness

Repetir hasta que el criterio sea satisfecho:

Para cada partícula i:

Actualizar V_i^t y X_i^t de acuerdo con las ecuaciones previas

Evaluar la función fitness f(X_i^t)

pbest$_i$ <- X_i^t si f(pbest$_i$) < f(X_i^t)

gbest <- X_i^t si f(gbest) < f(X_i^t)

Donde i es la iteración a través de las partículas del enjambre y j es la iteración a través de las dimensiones del espacio donde se busca la solución, esto se verá más claro en el pseudocódigo más adelante.

En esta práctica se emplearon las mismas funciones que para la practica 3, estas se verán en el siguiente apartado.

Funciones de para pruebas de optimización usadas para la práctica.

Función de Ackley:

Un dominio de dos dimensiones está definido por:

$$f(x,y) = -20\exp\left[-0.2\sqrt{0.5(x^2 + y^2)}\right] - \exp[0.5(cos2\pi x + cos2\pi y)] + e + 20$$

Esta función no convexa fue diseñada por David Ackley en 1987 para probar el rendimiento de algoritmos de optimización.

Los límites de evaluación del dominio son:

$$-5 \leq x, y \leq 5$$

El mínimo global optimo es:

$$f(0,0) = 0$$

La representación gráfica de la función es la siguiente:

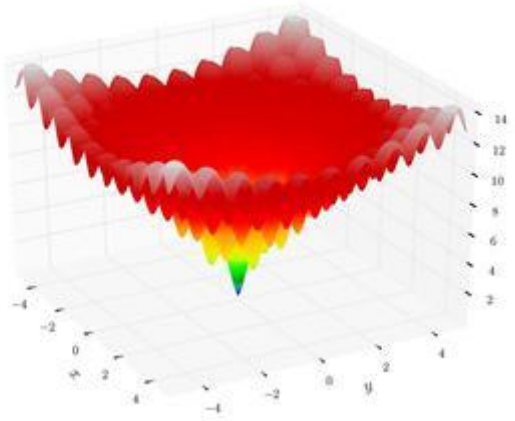

Figura 1. Grafica de la función de Ackley.

Función de Rastrigin:

En un dominio está definido por:

$$f(x, \ldots, x_n) = 10n + \sum_{i=1}^{n} [x_i^2 - 10cos(2\pi x_i)]$$

Donde n es el número de dimensiones.

Esta función no convexa fue diseñada por Rastrigin en 1974 para probar el rendimiento de algoritmos de optimización y fue propuesta originalmente para dos dimensiones, aunque posteriormente fue generalizada para n dimensiones.

Los límites de evaluación del dominio son:

$$x_i \in [-5.12, 5.12]$$

El mínimo global optimo es:

$$f(x^*) = 0 \; con \; x^* = (0, \dots, 0)$$

La representación gráfica de la función es la siguiente:

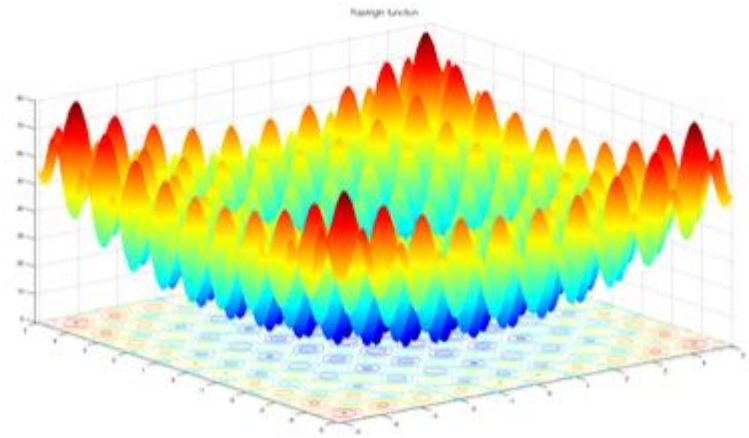

Figura 2. Grafica de la función de Rastrigin para 2 dimensiones.

Material y equipo

- El programa fue realizado en Google Colaboratory usando Python 3.9 como lenguaje.
- El sistema que se utilizó para acceder a Google Colaboratory fue una laptop de marca Lenovo, modelo Ideapad S540 con Ryzen 7 y 8 GB de RAM, con sistema operativo Windows 10.

Desarrollo de la practica

Función de Ackley

Para esta práctica se empleó el código visto en la sesión de laboratorio del mismo tema, pero aplicando algunas modificaciones para la aplicación de las funciones de Rastrigin y Ackley en 3 y 2 dimensiones respectivamente.

Para la función de Ackley, lo primero que hacemos es utilizar la función de Ackley como función para evaluar.

```
# Función a evaluar con el algoritmo
def ackley_function(x, y):
  return -20.0 * np.exp(-0.2 * np.sqrt(0.5 * (x**2 + y**2))) - \
  np.exp(0.5 * (np.cos(2 * np.pi * x)+np.cos(2 * np.pi * y))) + np.e + 20
```

Figura 3. Función de Ackley para el algoritmo de optimización.

El algoritmo toma en cuenta las dimensiones en las que se está buscando la respuesta más óptima, por lo que como tal no hay tantas modificaciones, las que se realizaron fueron porque la función (la función de Ackley en este caso) recibe los argumentos separados en vez de recibir una lista de argumentos. En las siguientes imágenes pueden observar algunos de estos cambios mencionados.

```
for i in range(0, n_particles):
    if ackley_function(X_lbest[i][0], X_lbest[i][1]) < ackley_function(X_gbest[0], X_gbest[1]):
        X_gbest = 1*X_lbest[i]
```

Figura 4. Cambio en la selección del gbest según la partícula más cercana al valor más óptimo de la función.

En la función de iteración también haremos un cambio similar.

```
# Update the particle velocity and position
for I in range(0, n_particles):
    for J in range(0, n_dimensions):
      R1 = np.random.rand()#uniform_random_number()
      R2 = np.random.rand()#uniform_random_number()
      V[I][J] = (weight*V[I][J]
                + C1*R1*(X_lbest[I][J] - X[I][J])
                + C2*R2*(X_gbest[J] - X[I][J]))
      X[I][J] = X[I][J] + V[I][J]
    if ackley_function(X[I][0], X[I][1]) < ackley_function(X_lbest[I][0], X_lbest[I][1]):
        X_lbest[I]=1*X[I]
        if ackley_function(X_lbest[I][0], X_lbest[I][1]) < ackley_function(X_gbest[0], X_gbest[1]):
            X_gbest=1*X_lbest[I]
```

Figura 5. Cambio en actualización de gbest en la iteración según la partícula más cercana al valor más optimo.

En cuanto a los resultados, usando los vectores de velocidad en la forma en que están creados originalmente, la inercia, la historia personal de la partícula y el seguimiento de la tendencia, de esta manera se conseguían resultados que variaban mucho, algunos eran muy cercanos al resultado esperado, pero otros se alejaban como el siguiente.

```
Generation 1900: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2000: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2100: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2200: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2300: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2400: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911

Generation 2500: Best particle in: [-0.98298172 -1.96362608]
gbest: 5.381897893937911
```

Figura 6. Mejor resultado después de 2500 iteraciones.

Después de esto se buscaron algunos cambios, como una mayor adición a la inercia.

```
# Loop until convergence, in this example a finite number of iterations chosen
weight=0.7 # Inercia
C1=0.2 # Historia personal

C2=0.1 # Seguir la tendencia que va marcando el lider

count+=1
```

Figura 7. Cambios en las variables de inercia y demás en las iteraciones.

Tenemos resultados más favorables, incluso si estos aun varían, son más cercanos al resultado esperado.

```
Generation 1900: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2000: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2100: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2200: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2300: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2400: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869

Generation 2500: Best particle in: [-9.52166544e-01 -9.32979481e-10]
gbest: 2.579927557029869
```

Figura 8. Resultados obtenidos después de los cambios realizados.

Tras eso hacemos otro cambio, esta vez en la velocidad inicial, esta será mucho mayor a la que se usaba regularmente, esta se aumenta considerablemente.

```
# Initialize the particle positions and their velocities
#Bias the initial population
X = lower_limit + 0.25*(upper_limit - lower_limit) * np.random.rand(n_particles, n_dimensions)
assert X.shape == (n_particles, n_dimensions)
# V = np.zeros(X.shape)
V = -(upper_limit - lower_limit) + 2*(upper_limit - lower_limit)*np.random.rand(n_particles, n_dimensions)
```

Figura 9. Cambios realizados en los vectores de velocidad para las partículas.

Podemos ver que los resultados mejoran considerablemente, incluso tomando en cuenta el factor del azar, en la mayoría de los intentos se consiguen resultados muy cercanos al más óptimo.

```
Generation 1900: Best particle in: [ 0.00029078 -0.00299211]
gbest: 0.008743478368284485

Generation 2000: Best particle in: [-0.00037653 -0.002261  ]
gbest: 0.006623045716285247

Generation 2100: Best particle in: [-0.00036994 -0.00224173]
gbest: 0.006563777382964986

Generation 2200: Best particle in: [-0.00036994 -0.00224173]
gbest: 0.006563777366725532

Generation 2300: Best particle in: [-0.00036994 -0.00224173]
gbest: 0.006563777366725532

Generation 2400: Best particle in: [-0.00036994 -0.00224173]
gbest: 0.006563777366725532

Generation 2500: Best particle in: [-0.00036994 -0.00224173]
gbest: 0.006563777366725532
```

Figura 10. Resultados obtenidos después de los cambios realizados.

Función de Rastrigin

Para la función de Rastrigin se hizo algo similar, ya considerando los cambios que utilizamos para la función de Ackley. Lo primero que se realizo fue el cambio de la función para evaluar por la función de Rastrigin.

```
# Función a evaluar con el algoritmo
def rastrigin_function(x, y, z):
  return 10*3 + (x**2 - 10*np.cos(2*np.pi*x)) + \
         (y**2 - 10*np.cos(2*np.pi*y)) + \
         (z**2 - 10*np.cos(2*np.pi*z))
```

Figura 11. Función de Rastrigin para la optimización de un espacio.

Posteriormente se hicieron los mismos cambios necesarios como los que se hicieron la función de Ackley, esto fue cambiar la función y sus parámetros ya que no recibe una lista de argumentos como tal.

```
for i in range(0, n_particles):
    if rastrigin_function(X_lbest[i][0], X_lbest[i][1], X_lbest[i][2]) < rastrigin_function(X_gbest[0], X_gbest[1], X_gbest[2]):
        X_gbest = 1*X_lbest[i]
```

Figura 12. Cambio en la selección del gbest según la partícula más apta evaluada en la función de Rastrigin.

El siguiente cambio hecho fue de igual manera en la función de iteración, cuando se evalúan nuevamente las partículas para decidir cuál de estas es la “mejor”.

```
# Update the particle velocity and position
for I in range(0, n_particles):
    for J in range(0, n_dimensions):
      R1 = np.random.rand()#uniform_random_number()
      R2 = np.random.rand()#uniform_random_number()
      V[I][J] = (weight*V[I][J]
                + C1*R1*(X_lbest[I][J] - X[I][J])
                + C2*R2*(X_gbest[J] - X[I][J]))
      X[I][J] = X[I][J] + V[I][J]
    if rastrigin_function(X[I][0], X[I][1], X[I][2]) < rastrigin_function(X_lbest[I][0], X_lbest[I][1], X_lbest[I][2]):
       X_lbest[I]=1*X[I]
       if rastrigin_function(X_lbest[I][0], X_lbest[I][1], X_lbest[I][2]) < rastrigin_function(X_gbest[0], X_gbest[1], X_gbest[2]):
           X_gbest=1*X_lbest[I]
```

Figura 13. Cambios en la función de iteración en la parte donde se actualizan las posiciones de las partículas y se actualiza cual es la partícula con el mejor resultado.

Después de esto probamos el funcionamiento de este algoritmo, pero con los cambios ya previamente realizados para la función de Ackley, es decir una velocidad mucho mayor, C1 = 0.2, C2 = 0.1 y la Inercia = 0.7, se consiguieron buenos resultados, pero en esta ocasión aun variaban bastante, en la siguiente figura se puede ver un resultado relativamente cercano al esperado.

```
Generation 1900: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2000: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2100: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2200: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2300: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2400: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498

Generation 2500: Best particle in: [-1.00384591e+00  1.63336795e-03 -1.98377947e+00]
gbest: 2.647443448162498
```

Figura 14. Resultados obtenidos aplicando de los cambios realizados.

Al aumentar a 20 el número de partículas que se usaran, obtenemos resultados consistentemente mejores, más cercanos al resultado más óptimo como el siguiente.

```
Generation 1900: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2000: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2100: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2200: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2300: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2400: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884

Generation 2500: Best particle in: [-0.00617568  0.03187217 -0.99585981]
gbest: 1.2035501087906884
```

Figura 15. Resultados obtenidos empleando 20 partículas.

Estos números pueden mejorar, con el aumento de partículas usadas, el aumento de velocidad y un mayor número de iteraciones, es interesante ver como estos cambios llevan a un mejor resultado, pero hay que considerar que también la complejidad temporal del algoritmo aumenta.

Conclusiones y recomendaciones

Esta práctica fue bastante interesante de realizar ya teniendo la perspectiva del uso de los algoritmos genéticos para trabajar las mismas funciones de optimización que se igual se usaron en esta práctica, considero que la implementación de los algoritmos de optimización por enjambre de partículas fue un poco más sencilla de implementar, pero algunas comparativas que pude notar es que a veces eran más propensos a quedarse atorados en un mínimo local o si las variables se cambiaban mucho para que hubiera mayor exploración, estos se dispersaban más de lo necesario y era un poco más complicado que encontraran el mínimo global, esto no necesariamente implica que uno sea más favorable que el otro para resolver esta clase problemas, sino que son características que hay que considerar al implementarlos. En si esta práctica requería estar haciendo diferentes pruebas para encontrar este mínimo global, pero considero que fue bastante ilustrativa en cómo funciona esta clase de algoritmo.

Bibliografía

de Almeida, B. S. G. (2019, 3 diciembre). *Particle Swarm Optimization: A Powerful Technique for Solving Engineering Problems*. IntechOpen. Recuperado 22 de octubre de 2021, de https://www.intechopen.com/chapters/69586

Tam, A. (2021, 11 octubre). *A Gentle Introduction to Particle Swarm Optimization*. Machine Learning Mastery. Recuperado 22 de octubre de 2021, de https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/

Carpitella, S. (s. f.). *Funciones test para optimización mono-objetivo*. Universidad Politecnica de Valencia. Recuperado 4 de octubre de 2021, de https://m.riunet.upv.es/bitstream/handle/10251/105210/Izquierdo%3BCARPITELLA%20-%20Funciones%20test%20para%20optimizaci%C3%B3n%20mono-objetivo.pdf?sequence=1&isAllowed=y