ALGORITMO EVOLUTIVO

Explicación de como resuelve Funciona el algoritmo:

1. Crea una población inicial de individuos (soluciones candidatas) aleatorios. Cada individuo sería una lista de vértices, de tamaño k, que representa un subconjunto de vértices del grafo.
2. Evalúa cada individuo de la población utilizando una función de aptitud que cuente el número de aristas entre los vértices del subconjunto.
3. Selecciona dos métodos de selección diferentes para seleccionar a los individuos más aptos para reproducirse. Por ejemplo, podrías utilizar el método de selección por torneo y el método de selección por ruleta.
4. Se aplica dos operadores de recombinación diferentes a pares de individuos seleccionados para crear nuevos individuos. Por ejemplo, podrías utilizar el operador de recombinación por un punto y el operador de recombinación uniforme.
5. Aplica dos operadores de mutación diferentes a algunos de los nuevos individuos para introducir variabilidad en la población. Por ejemplo, podrías utilizar el operador de mutación por intercambio y el operador de mutación por inserción.
6. Si alguno de los nuevos individuos es inválido (por ejemplo, contiene vértices repetidos o más de k vértices), Se repara o penaliza su aptitud de alguna manera.
7. Reemplaza a algunos de los individuos de la población original con los nuevos individuos.
8. Repite los pasos 3 a 7 hasta que se alcance el criterio de parada (por ejemplo, un número máximo de generaciones o una solución óptima).

TIPO DE ALGORITMO

Este código implementa un algoritmo evolutivo, que es un tipo de algoritmo de optimización inspirado en el proceso de evolución de las especies en la naturaleza.

Los algoritmos evolutivos consisten en una población de individuos que representan posibles soluciones a un problema. A través de un proceso iterativo, se seleccionan individuos de la población para reproducirse y crear una nueva generación de individuos. Estos individuos suelen tener pequeñas variaciones respecto a sus progenitores, y estas variaciones se consiguen aplicando operadores de recombinación y mutación. Los individuos se evalúan mediante una función de aptitud, que mide su calidad como solución al problema. A medida que se van generando nuevas generaciones, se espera que la aptitud media de la población aumente y que se encuentren individuos cada vez mejores.

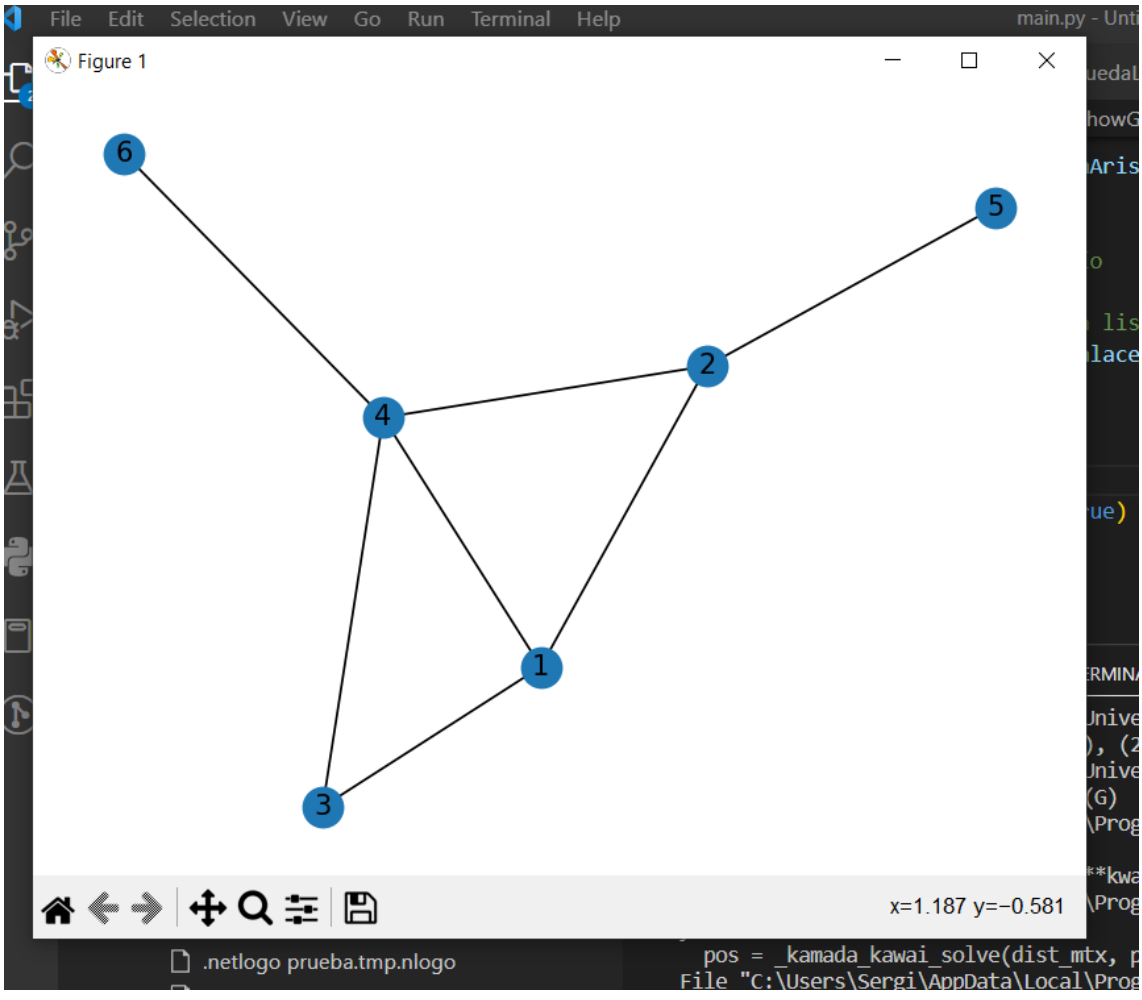
En el caso de este código en particular, se está tratando de encontrar un subconjunto de vértices de un grafo que maximice el número de aristas dentro de ese subconjunto, siempre y cuando al menos dos vecinos diferentes sean explorados. Para ello, cada individuo de la población representa un subconjunto de vértices, y la función de aptitud cuenta el número de aristas que hay dentro del subconjunto. Luego, se aplican dos operadores de recombinación y dos operadores de mutación diferentes para crear una nueva generación a partir de los individuos seleccionados. Finalmente, se obtiene el mejor individuo de la población.

Pruebas

Probaremos con los datos del ejemplo del .pdf.

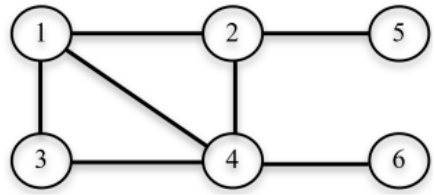
- La solución mas optima debería ser:
  - S2 = {1,2,3,4} -> Number of Edges = 5
- Podemos ver que el grafo del ejemplo es el mismo que el que estamos utilizando mediante la siguiente función:

```python
def showGraph(listaEnlaces):
        # Crea un grafo vacío
    G = nx.Graph()
    # Añade los enlaces a la lista de enlaces del grafo
    G.add_edges_from(listaEnlaces)
    # Dibuja el grafo
    nx.draw(G, with_labels=True)
    # Muestra el gráfico
    plt.show()
```



Example:
Consider a graph with 6 vertices and 7 arcs, represented below (***test.txt*** in Moodle). The image on the right illustrates the graph. The goal is to find a solution of the problem for k=4.

```
k 4
p edge 6 7
e 1 2
e 1 3
e 1 4
e 2 4
e 2 5
e 3 4
e 4 6
```



Consider the three solutions (there will be others):
S1 = {1, 2, 5, 6} - number of edges = 2
S2 = {1, 2, 3, 4} - number of edges = 5
S3 = {2, 3, 5, 6} - number of edges = 1

```python
def evolutiveAlgorithm():
    # Parámetros del algoritmo
    vertices = 6
    nAristas = 7
    kValue = 4
    listaEnlaces = [(1, 2), (1, 3), (1, 4), (2, 4), (2, 5), (3, 4), (4, 6)]
    poblacion_inicial = 10
    max_generaciones = 100
    tasa_mutacion = 0.1
```

Y el resultado es:

```
PS C:\Users\Sergi\Desktop\code\Universidad\Erasmus\IA> & C:/User
i/Desktop/code/Universidad/Erasmus/IA/p2/Entregable/main.py
[4, 2, 3, 1] Number of Edges =  5
PS C:\Users\Sergi\Desktop\code\Universidad\Erasmus\IA>
```

- (Este resultado puede variar dependiendo la iteración del algoritmo):

```
i/Desktop/code/Universidad/Erasmus/IA/p2/Entregable/main.py
[5, 4, 2, 1] Number of Edges =  4
```