AI

ALGORITHM

Sergio Gonzalez Martinez

Lucia Arnaiz

content

AIM

The objective of this text is to create an algorithm that, given a graph (a set of points connected to each other by links) and an integer k, finds a subset of k points of the graph that are connected to each other by the greatest possible number of links. It is a maximization problem, which means that we want to find the solution that has the largest possible value.

ALGORITHM (GREEDY)

Explanation of how the algorithm solves and works:

1. Sorts the list of tuples in descending order by the number of neighbors.
2. Select the k vertices with the most neighbors. If there is a tie, select the vertices with the most neighbors in order of appearance in the list.
3. Counts the number of edges between the selected vertices.

ALGORITHM TYPE

This algorithm is a heuristic solution to the problem of finding a subset of vertices in a graph that maximizes the number of edges between them.

This algorithm uses a greedy approach , that is, it makes local optimal decisions at each step in the hope of reaching a global optimal solution.

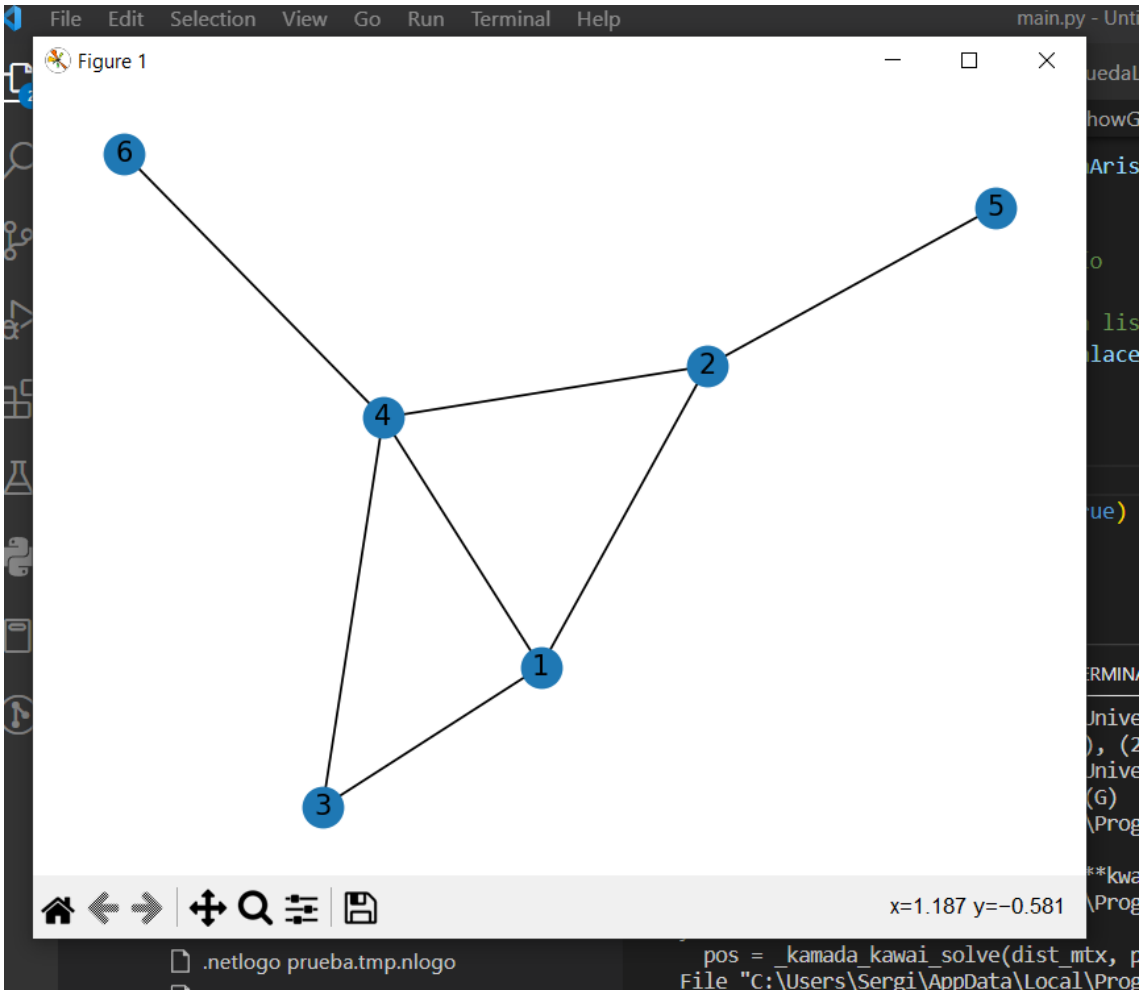
The optimal local decision **is to select at each step the vertex with the most neighbors** , since it has the best chance of having more edges with other vertices.

Testing

We will try with the data from the example of the . pdf .

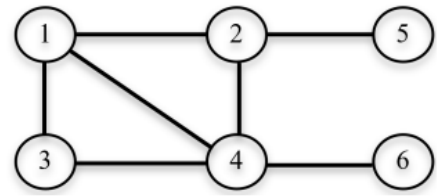
- most optimal solution should be:
  - S2 = {1,2,3,4} -> Number of edges = 5
- We can see that the example graph is the same as the one we are using using the following function:

```
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

```
def AlgoritmoGreedy(kValue,vertices,nAristas,listaEnlaces):
```

We add the values of the example:

```
AlgoritmoGreedy(4,6,7,[(1, 2), (1, 3), (1, 4), (2, 4), (2, 5), (3, 4), (4, 6)])
```

And the result is:

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

- (This result may vary depending on the iteration of the algorithm):

EVOLUTIONARY ALGORITHM

Explanation of how the algorithm solves and works:

1. Creates an initial population of random individuals (candidate solutions). Each individual would be a list of vertices, of size k, that represents a subset of vertices of the graph.
2. Evaluate each individual in the population using a fitness function that counts the number of edges between the vertices of the subset.
3. Select two different selection methods to select the most fit individuals to reproduce. For example, you could use the tournament pick method and the roulette pick method.
4. Two different recombination operators are applied to pairs of selected individuals to create new individuals.

5. It applies two different mutation operators to some of the new individuals to introduce variability into the population.
6. If any of the new individuals is invalid (for example, it contains repeated vertices or more than k vertices), its fitness is repaired or penalized in some way.
7. Replaces some of the individuals from the original population with the new individuals.
8. Repeat steps 3 through 7 until the stopping criterion is reached (for example, a maximum number of generations or an optimal solution).

ALGORITHM TYPE

This code implements an evolutionary algorithm, which is a type of optimization algorithm inspired by the process of evolution of species in nature.

Evolutionary algorithms consist of a population of individuals that represent possible solutions to a problem. Through an iterative process, individuals are selected from the population to reproduce and create a new generation of individuals. These individuals usually have small variations with respect to their parents, and these variations are achieved by applying recombination and mutation operators. Individuals are evaluated by a fitness function, which measures their quality as a solution to the problem , that is, the number of links in a subset. As new generations are generated, it is expected that the average fitness of the population will increase and that better and better individuals will be found.

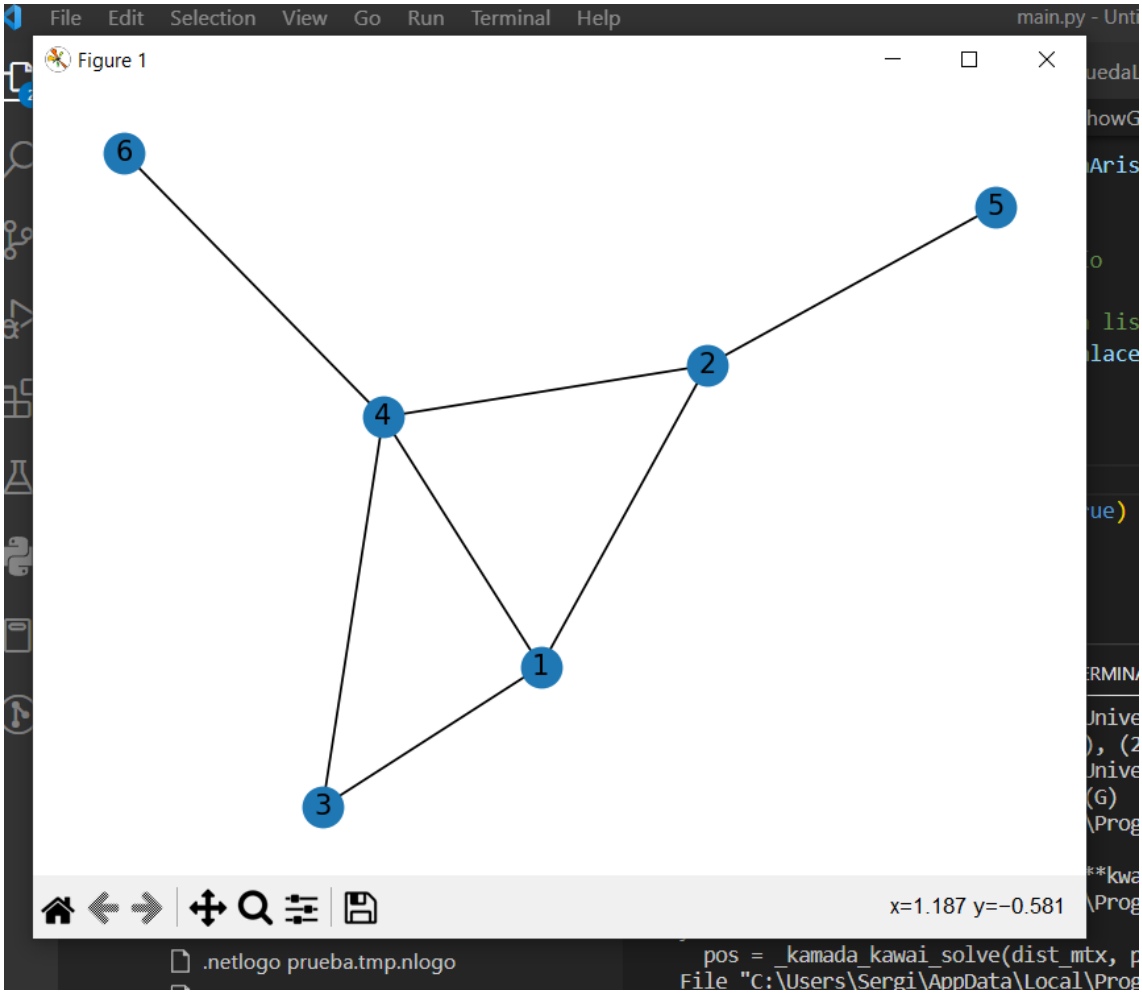
In the case of this particular code, it is trying to find a subset of vertices of a graph that maximizes the number of edges within that subset, as long as at least two different neighbors are explored. To do this, each individual in the population represents a subset of vertices, and the fitness function counts the number of edges within the subset. Two different recombination operators and two different mutation operators are then applied to create a new generation from the selected individuals. Finally, the best individual in the population is obtained.

Testing

We will try with the data from the example of the . pdf .

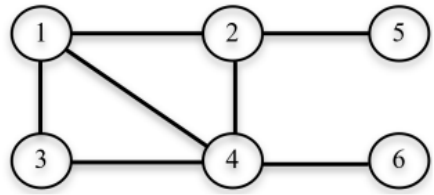
- most optimal solution should be:
    - S2 = {1,2,3,4} -> Number of edges = 5
- We can see that the example graph is the same as the one we are using using the following function:

```
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
```

And the result is:

```
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>
```

- (This result may vary depending on the iteration of the algorithm):

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

STOCHASTIC HILL CLIMBING ALGORITHM

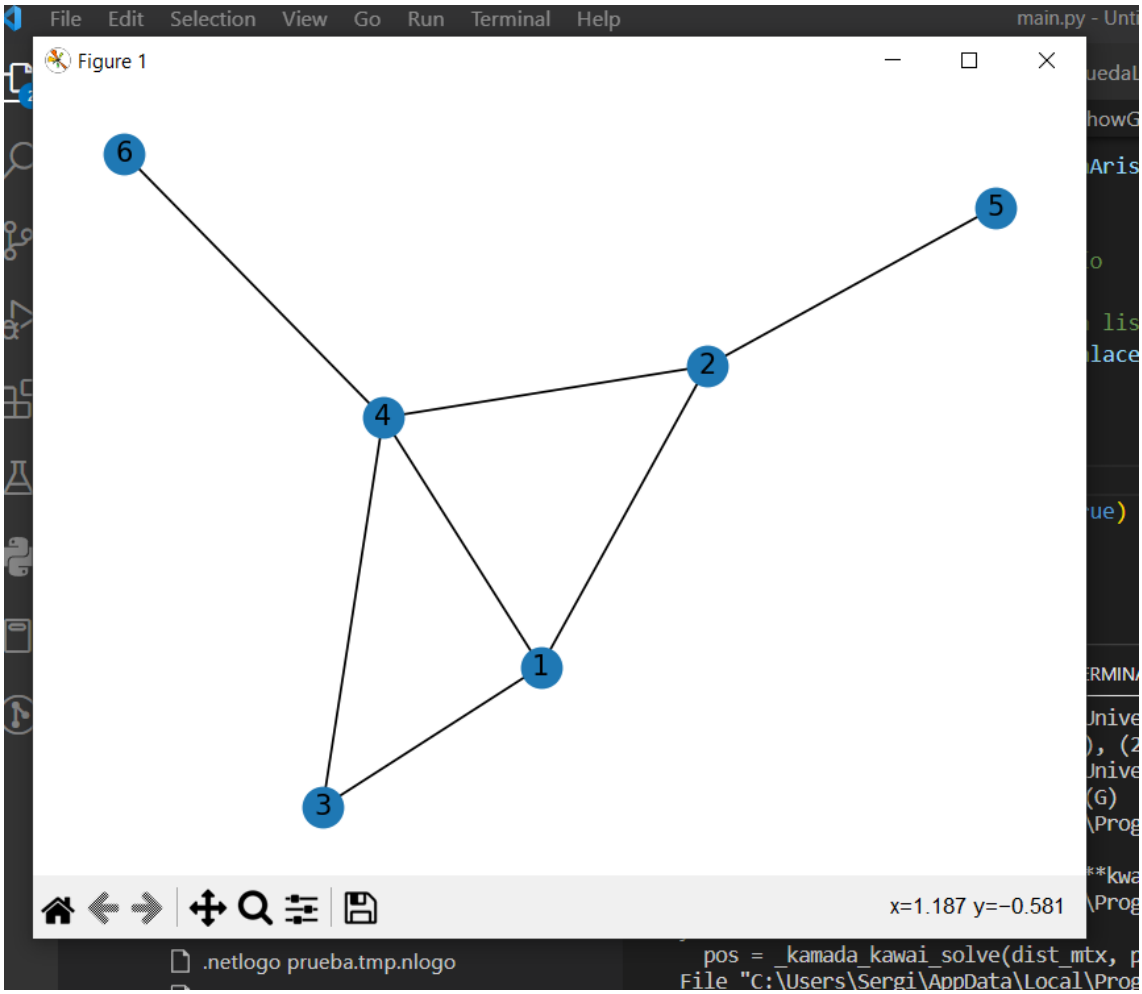
This algorithm is an implementation of the stochastic hill climbing algorithm. It is a type of local search algorithm used to find an optimal (or very close to optimal) solution for a given problem. The algorithm starts with an initial solution, and then iteratively makes small changes to the current solution in an attempt to improve it. If a change results in a better solution, it becomes the new solution and the process continues. If there is no way to improve the current solution, the algorithm terminates and returns the final solution.

Testing

We will try with the data from the example of the . pdf .

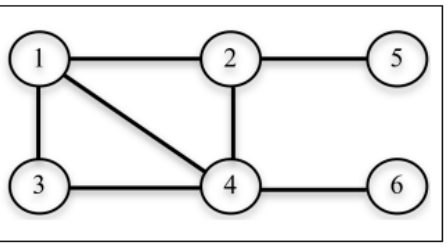
- most optimal solution should be:
  - S2 = {1,2,3,4} -> Number of edges = 5
- We can see that the example graph is the same as the one we are using using the following function:

```
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

And the result is:

```
**********          ALGORITMO Hibrido ESTOCASTICO - EVOLUTIVO      **********
[4, 3, 2, 1] -> Number of Edges =  5
```

HYBRID GREEDY-EVOLUTIONARY ALGORITHM

Explanation of how the algorithm solves and works:

This hybrid algorithm combines elements of an evolutionary algorithm and a greedy local search algorithm .

The evolutionary algorithm is used to find a good solution to a problem by using selection, recombination, and mutation techniques. In this particular case, two selection methods (tournament selection and roulette selection) are used to choose the individuals to reproduce, and two recombination operators are applied (one-point recombination and uniform recombination) to produce two groups of individuals. children. Two mutation operators (exchange mutation and insertion mutation) are then applied to each of the two sets of children.

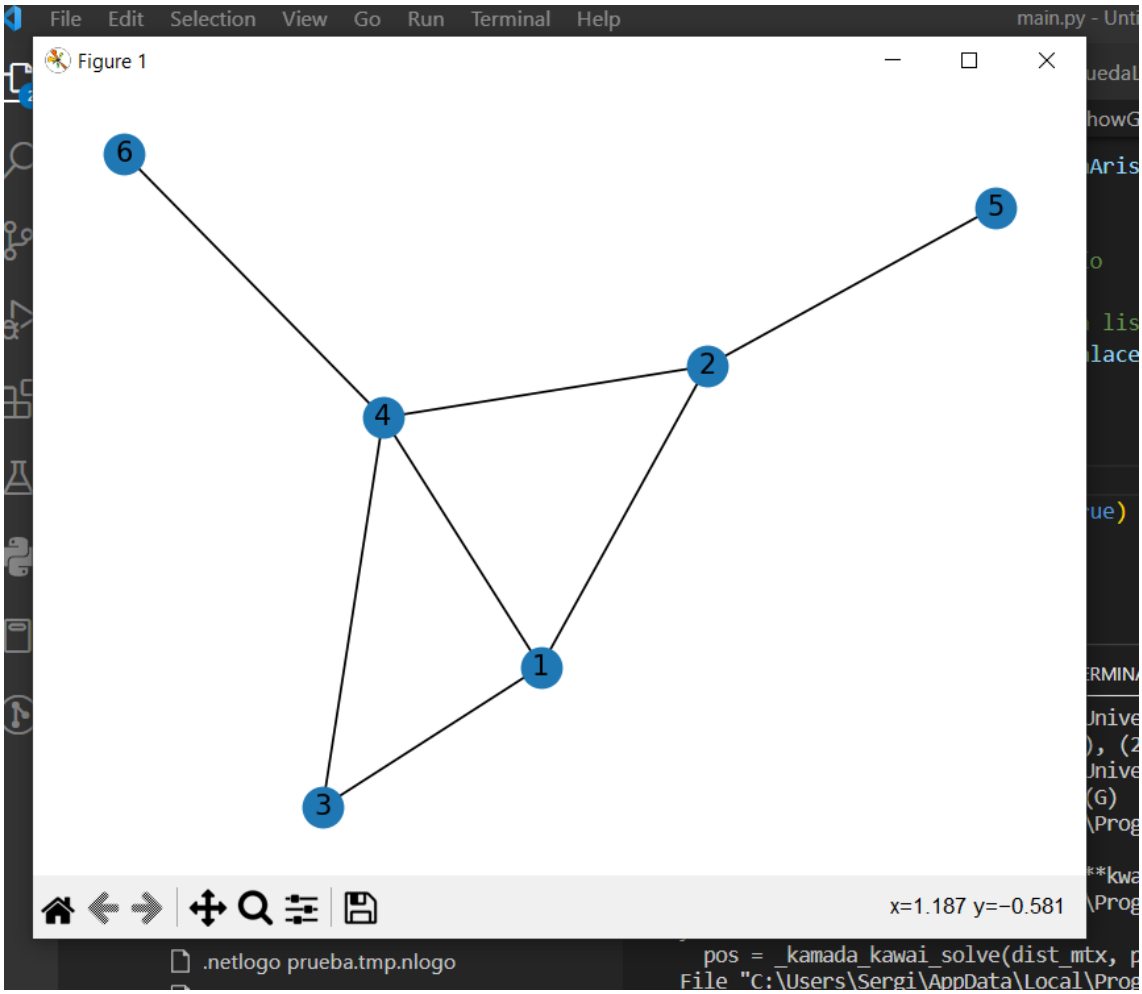
greedy local search algorithm uses a heuristic to find an optimal solution in a neighborhood of solutions.

Testing

We will try with the data from the example of the . pdf .

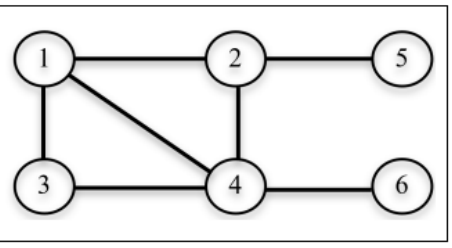
- most optimal solution should be:
  - S2 = {1,2,3,4} -> Number of edges = 5
- We can see that the example graph is the same as the one we are using using the following function:

```
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
```

And the result is:

```
[4, 2, 3, 1] Number of Edges =  5
PS C:\Users\Sergi\Desktop\code\Universidad\Erasmus\IA>
```

- (This result may vary depending on the iteration of the algorithm):

```
[5, 4, 2, 1] Number of Edges =  4
```

HYBRID EVOLUTIONARY STOCHASTIC ALGORITHM

Explanation of how it solves The algorithm works:

This hybrid algorithm combines the stochastic hill-climbing algorithm and the evolutionary algorithm to find an optimal solution to obtain the subset with the largest number of edges. The stochastic hill-climbing algorithm is used to generate initial solutions, while the evolutionary algorithm is used to improve those initial solutions.

In each iteration of the evolutionary algorithm, individuals to breed are first selected using a selection method, such as tournament selection or roulette selection. The breeding and mutation operators are then applied to the selected individuals to create new individuals (called offspring). Next, the fitness of the offspring is evaluated and the fittest individual is selected as the current new solution.

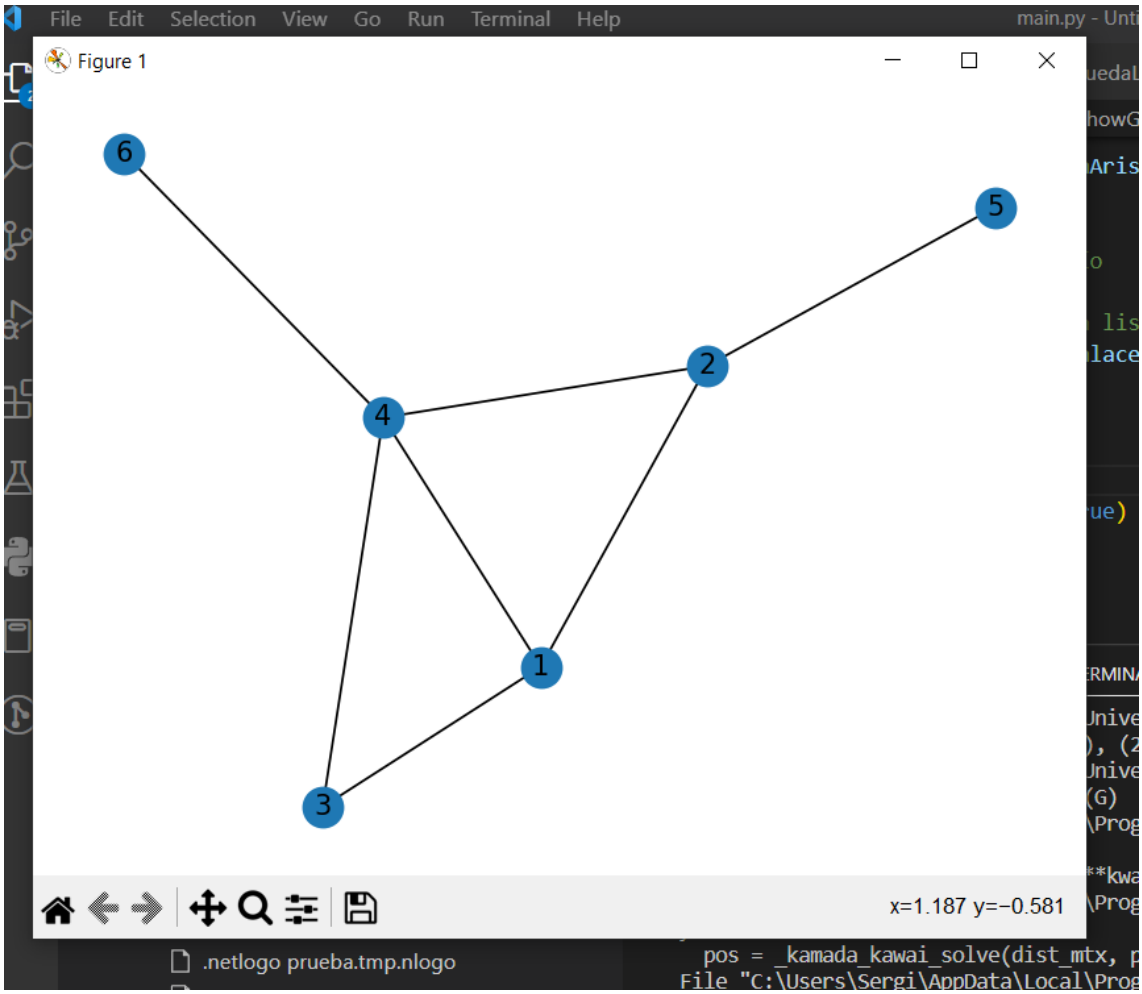
One of the advantages of using a hybrid algorithm is that you can take advantage of the strengths of each individual algorithm and minimize their weaknesses. In this case, the stochastic hill-climbing algorithm is fast and can find near-optimal solutions, while the evolutionary algorithm is capable of exploring a larger solution space and finding optimal solutions. By combining both algorithms, we can get a final solution that has a good combination of speed and optimization.

Testing

We will try with the data from the example of the . pdf .

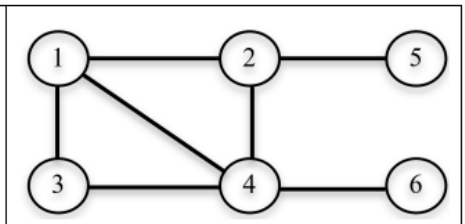
- most optimal solution should be:
  - S2 = {1,2,3,4} -> Number of edges = 5
- We can see that the example graph is the same as the one we are using using the following function:

```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

And the result is:

```
**********          ALGORITMO Hibrido ESTOCASTICO - EVOLUTIVO      **********
[4, 3, 2, 1] -> Number of Edges =  5
```

- (This result may vary depending on the iteration of the algorithm):

EVOLUTIONARY STOCHASTIC HYBRID ALGORITHM 2

The hybrid stochastic evolutionary2 algorithm starts by generating an initial solution using the stochastic hill-climbing algorithm and then uses an evolutionary process to improve that solution iteratively over several generations.

STUDY

Best Study Solutions:

File	Vertices	Edges	K	Best solution
teste.txt	6	7	4	5
file1.txt	28	210	8	21
file2.txt	64	704	7	16
file3.txt	70	1855	16	112
file4.txt	200	1534	14	79
file5.txt	500	4459	15	98

Greedy algorithm :

We can carry out a study with the number of maximum iterations and the number of maximum interactions WITHOUT improvement:

- We can see that: This type of algorithm is not very effective, instead it is the fastest of all due to its short execution time. We can also observe that the first result with the maximum number of interactions without improvement and with improvement is always the most efficient since there is no variation between the results. We can conclude that this greedy algorithm is not EFFICIENT in large graphs, instead it is in small graphs like the file teste.txt if it is.
- The best result is with:
  - File1.txt
    - Iterations : 100
    - MaxIteration No improvement: 100
    - Result: [8, 7, 6, 5, 4, 3, 2, 1] -> n: 9
  - file2.txt
    - Iterations : 100
    - MaxIteration No improvement: 100
    - Result: [] -> n: 0
  - File3.txt

        - Iterations : 100
        - MaxIteration No improvement: 100
        - Result: (16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1) -> n: 57
    - File4.txt
        - Iterations : 100
        - MaxIteration No improvement: 100
        - Result: (39, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2) -> n: 14
    - File5.txt
        - Iterations : 100
        - MaxIteration No improvement: 100
        - Result: (16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2) -> n: 14

Stochastic Hill Climb Algorithm:

We can perform a study with the number of maximum iterations, since the number of maximum iterations will be the number of neighborhoods that the algorithm visits:

- We can see that:
    - The results are very good, the difference with the best solution is minimal, therefore we can conclude that the algorithm is very efficient and gives what it promises.
    - It tends to find the most efficient solution in the first few iterations.
- The best result is with:
    - File1.txt
        - Iterations : 3200
        - Result: (26, 24, 21, 20, 12, 6, 4, 1) -> n = 20
    - file2.txt
        - Iterations : 100
        - Result: (63, 50, 42, 39, 29, 20, 9) -> n = 15

- File3.txt
  - Iterations : 20000
  - Result: (64, 60, 53, 51, 46, 43, 38, 35, 29, 28, 25, 20, 16, 11, 9, 3) -> n = 112
- File4.txt
  - Iterations : 200
  - Result: (195, 159, 158, 122, 121, 86, 85, 84, 49, 48, 47, 12, 11, 10) -> n = 79
- File5.txt
  - Iterations : 100
  - Result: (331, 330, 252, 251, 250, 172, 171, 170, 91, 90, 89, 13, 12, 11, 10) -> n = 93

Evolutionary Algorithm :

We can carry out a study with the number of maximum generations, the initial population and the mutation rate.

- We can see that:
  - In the first three files the Algorithm is EFFICIENT, since the distance of the solution obtained with the ideal distance is minimal.
  - In files 4 and 5 the algorithm loses efficiency and the results are not close to the desired ones.
  - It tends to find the best solution with a higher generation limit .
  - No significant changes in the population and mutation rate can be observed.

Tends to find the best solution with a higher maximum generation

- The best result is with:
  - File1.txt
    - Max Generations: 100
    - Starting population: 10
    - Mutation rate: 0.3
    - Result: (27, 23, 20, 19, 13, 8, 4, 2) -> Number of edges = 20
  - file2.txt
    - Max Generations: 500
    - Starting population: 16
    - Mutation rate: 0.1
    - Result: (56, 42, 39, 29, 20, 14, 3) -> Number of edges = 15
  - File3.txt
    - Max Generations: 500
    - Starting population: 16
    - Mutation rate: 0.1
    - Result: (67, 65, 61, 56, 49, 44, 42, 35, 34, 21, 20, 17, 10, 9, 4, 1) -> Number of edges = 104
  - File4.txt

        - Max Generations: 500
        - Starting population: 16
        - Mutation rate: 0.3
        - Result: (197, 196, 195, 195, 194, 123, 121, 48, 13, 12, 11, 10, 9, 8) -> Number of edges = 39
    - File5.txt
        - Max Generations: 500
        - Starting population: 20
        - Mutation rate: 0.1
        - Result: (483, 482, 411, 402, 401, 325, 322, 241, 82, 10, 9, 4, 3, 2, 1) -> Number of edges = 44

Algorithm 1:

We can perform a study with the number of maximum iterations, since the number of maximum iterations will be the number of neighborhoods that the algorithm visits, the initial population, and the number of maximum generations.

- We can see that:
  - The results of the hybrid algorithm are almost perfect, obtaining the ideal result in 3/5 files.
  - We can see that most of the time it is not necessary to increase the initial population and generations to obtain a better solution.
  - We can observe that due to the efficiency of this algorithm it is not necessary to increase the interactions for the study of these files.
- The best result is with:
  - File1.txt
    - Max Generations: 100
    - Starting population: 10
    - Iterations: 100
    - Result: (27, 22, 19, 17, 14, 9, 7, 3) -> Number of edges = 20
  - file2.txt
    - Max Generations: 100
    - Starting population: 10
    - Iterations: 100
    - Result: (52, 37, 35, 31, 25, 16, 6) -> Number of edges = 15
  - File3.txt
    - Max Generations: 500
    - Starting population: 10
    - Iterations:100
    - Result: (70, 59, 58, 50, 47, 46, 45, 40, 31, 26, 24, 21, 17, 13, 12, 1) -> Number of edges = 112

- File4.txt
  - Max Generations:1000
  - Starting population: 10
  - Iterations:100
  - Result : (196, 195, 160, 159, 158, 122, 121, 85, 84, 49, 48, 47, 11, 10) -> Number of Edges = 79
- File5.txt
  - Max Generations:100
  - Starting population: 20
  - Iterations:500
  - Result: (496, 495, 417, 416, 415, 336, 335, 256, 255, 176, 175, 96, 95, 16, 15) -> Number of edges = 98

Algorithm 1:

This algorithm uses the same parameters as the previous algorithm.

- We can see that:
  - The ideal result is achieved in 2/5 files and the other results are close to the ideal solution, therefore we can say that the result is very good.
  - It is observed that it is not necessary to increase in most cases the initial population in most cases to have a good solution.
  - No patterns are seen among the other variables to draw an observation.
- The best result is with:
  - File1.txt
    - Max Generations:100
    - Starting population: 10
    - Iterations:100
    - Result: (25, 24, 17, 16, 13, 12, 10, 7) -> Number of edges = 20
  - file2.txt
    - Max Generations:500
    - Starting population: 10
    - Iterations:100
    - Result: (60, 45, 33, 27, 22, 8, 3) -> Number of edges = 15
  - File3.txt
    - Max Generations: 1000
    - Starting population: 10
    - Iterations: 1000
    - Result: (60, 57, 52, 51, 46, 45, 43, 42, 34, 29, 28, 25, 20, 14, 11, 1) -> Number of edges = 110
  - File4.txt
    - Max Generations: 1000
    - Starting population: 10
    - Iterations: 100

        - Result: (195, 194, 158, 157, 121, 120, 84, 83, 82, 47, 46, 45, 10, 9) -> Number of edges = 79
    - File5.txt
        - Max Generations: 1000
        - Starting population: 20
        - Iterations: 500
        - Result: (494, 493, 414, 413, 334, 333, 254, 253, 175, 174, 173, 94, 93, 14, 13) -> Number of edges = 98

Difference between the two hybrid algorithms:

- There are no notable differences between the two algorithms since the result of both is good.
- In the first algorithm the ideal result was achieved in 3/5 files and in the second algorithm it was achieved in 2/5
- It could be said that from the results obtained algorithm 1 is more efficient than algorithm 2.

Note : No studies were performed on the hybrid greedy and evolutionary algorithm since the results of the local algorithm were not close to the desired ones.