TOOL_NAME

Vasileios Vlachos, Theodoros Siklafidis

Abstract

Graphs are useful data structures capable of efficiently representing a variety of technological and social networks. They are therefore utilized in simulation-based studies of new algorithms and protocols. Inspired by **NGCE - Network Graphs for Computer Epidemiologists** [1], we present the TOOL_NAME, an easy to use graph generator, analyzer and visualizer that produces structures for the study of complex computer networks.

1. Introduction

TOOL_NAME is a tool for creation, analysis and last but not least visualization in various graphs. It is written in **Python** [2] and provides a friendly **graphical user interface (GUI)** [3] developed with the known Python module **Tkinter** [4]. Initially the user is able to start exploring the different graph types and their behavior by generating them in the **Generate sector** [5] and trying different seed values provided by the **random** [6] module of Python. The user is also given an interesting choice as far as the structure of the generated graph is concerned, between **Adjacency Matrix** [7] or **Adjacency List** [8]. Furthermore, reaching the **Analyze sector** [9] the user is given various options. He can analyze the last generated graph either this graph was generated with an adjacency matrix or an adjacency list, and choose the available metrics (**Geodesic Paths** [10], **Closeness Centrality** [11],

Betweenness Centrality [12]) that he would like to calculate for a particular graph. Finally, we have the **Visualize sector** [13] where the user can visualize a generated graph with a tool called **Pajek** [14], a web platform called **Plotly** [15] or a python module named **matplotlib** [16].

2. Network Graph Topologies

2.1 Homogeneous Graphs

Homogeneous or fully connected graphs were for many years the epidemiologists' preferred choice for describing the spread of infectious diseases. This topology has been recently adapted to model the growth of computer viruses and worms. TOOL_NAME supports homogeneous graphs because they offer significant advantages. First, analytical mathematical models can be easily applied to them; second, they provide a good abstraction of very large networks when the majority of the susceptible hosts are accessible from an infectious agent and third, performing simulations using a homogeneous graph and comparing their outcomes with the mathematical analytical results is an excellent way to ensure that implementation details did not corrupt the simulation model.

Homogeneous graphs can be built as follows:

1) Start with N vertices and no edges.
2) Connect each vertex with all the other vertices.

A homogeneous graph with N nodes will always end up with

$(N * (N - 1)) / 2$ edges.

2.2 Random Graphs

Until recently, it was believed that random graphs provide a good approximation to very large networks such as the Internet. Barabási et al proved however, that the connectivity of the Internet, along with that of many other technical and social networks, obeys a power law distribution forming scale-free graphs. Prior to these findings, a large number of simulations that investigated the spread of malicious code, had been performed on random graphs. Due to this fact, and, more importantly, in order to allow the comparison between older and current studies, we decided to include them in our tool. Since numerous algorithms have been proposed to construct random graphs, we selected to implement the most widely adopted one, in particular the Erdős–Rényi algorithm or the Gilbert algorithm according to others.

The algorithm works as follows:

1) Start with a graph with N nodes and no edges.
2) Connect each pair of two nodes with probability Per. This results in an Erdős–Rényi graph.

The shape of the connectivity distribution however is highly dependent on the value of the Per probability.

2.3 Scale-Free Graphs

Scale-free structures exist in a stunning range of heterogeneous systems ranging from biological and social to purely technological networks such as the World Wide Web (WWW), the physical connectivity of the Internet or the network of people connected by e-mail. Studies have explored the spread of malicious code in scale-free computer networks with interesting but conflicting results while investigated the resilience of the Internet to random breakdowns. We assume that, in light of these recent evidences, simulation research will increasingly be based on scale-free topologies and thus we support them in the TOOL_NAME tool. Barabási and Albert demonstrated that the creation of scale-free networks should include two definitive characteristics: Incremental growth and preferential connectivity. **Incremental growth** is the process of adding new nodes to an existing graph. **Preferential connectivity** or **preferential attachment** describes the tendency of a newly added node to be connected with highly connected nodes. The elimination of either of these properties lead to graphs with temporal scale-free characteristics. Under certain circumstances it is possible to construct scale-free graphs without enforcing growth and preferential connectivity simultaneously. We felt that the inclusion of a flexible model that allows the experimentation with either of these modes or a combination of them would leverage the development of a larger variety of scale-free graphs.

Preferential connectivity: In order to construct a BA (Barabási and Albert) graph working with a fixed number of nodes without the incremental growth property, but with the preferential connectivity, we have to use a limited number of edges.

The algorithm consists of the following phases:

1) If the number of edges is less than the number of nodes, select randomly a vertex and connect it with probability $P(k_i) = \frac{ki}{\sum j\ kj}$ to the vertex.
2) Repeat step 1. If the number of edges is approximately equal to the number of nodes the constructed graph exhibits scale-free characteristics.

Incremental growth: One, but not sufficient ingredient, of the scale-free networks is the incremental growth property, but without the preferential connectivity option; as new nodes are added to the graph the scale-free nature of the network diminishes. We thought that it would be best, if we enabled the construction of graphs based solely on the incremental growth for further study and experimentation.

The algorithm has the following structure:

1) Create a pool K and add to it mo initial nodes.

2) Create a pool L of all the other nodes.

3) Remove randomly a node i from the pool L and connect it to a randomly chosen node from the pool K.

4) Add node i to the pool K.

5) While the pool L is not empty, repeat step 3. It is also possible to connect a node with $m > 1$ other nodes from pool K as long as the size of the pool K is larger than m.

Complete model: By combining both of the incremental growth and preferential connectivity properties our system will approximate the BA model as closely as possible. The algorithm is constituted by two major parts.

For the first mo nodes:

1) Create a pool K and add to it mo initial nodes.

2) Create a pool L of all the other nodes.

3) Remove randomly a node l from the pool L and connect it to a randomly chosen node from the pool K.

4) Add node l to the pool K.

And for the rest of the nodes, we follow closely the BA's algorithm:

1) Remove randomly a node i from the pool L.

2) Select randomly a vertex from the pool K and connect it with probability $P(k_i) = \frac{ki}{\sum j\ kj}$ to the vertex i.

3) Add i to the pool K.

The first part of the algorithm is arbitrary since Barabási and Albert, did not describe how to connect the mo + 1 node. To verify that the produced graphs obey the power law distribution necessary to characterize a graph as scale-free, we developed scripts to parse the constructed graph and extract the connectivity probabilities and create a distribution with matplotlib.

2.4 Random Graphs with Fixed Connectivity

Random graphs with fixed connectivity constitute a nontrivial network topology that is often encountered in grid systems. Considering the significant importance of such systems and the fact that older studies have been based on them, we also included these structures in our TOOL_NAME tool. We implemented a custom algorithm for the creation of random graphs with fixed connectivity. In our approach, the user chooses the number of nodes and the number of edges each node should contain.

Then the algorithm works as follows:

1) Create an unconnected graph with N nodes.

2) Add all nodes to a pool L.

3) Pick randomly a vertex v and a vertex u from the pool L, if the pool L has at least two elements, else exit.

4) Connect vertex v with vertex u and decrease their available connectivity by one.

5) If vertex's v or vertex's u available connectivity is 0, remove it from the pool L.

6) Repeat step 3.

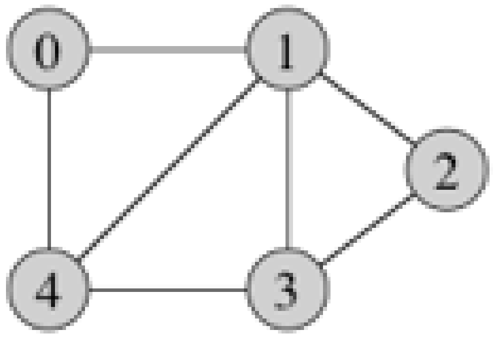
2.5 Custom Graphs

Sometimes it is necessary to measure the effects of various algorithms in nonstandard graphs. We therefore added an option to our system that gives the ability to an experienced user to create non-typical graphs with custom properties based either on the random graph algorithm or on the scale-free algorithm.

3. Generate sector

Starting with generating a graph the user can explore the various types of graphs provided by the TOOL_NAME and understand better their characteristics and behaviors. After the user chooses a graph type that he wants to create he gets to choose between the adjacency matrix or adjacency list.

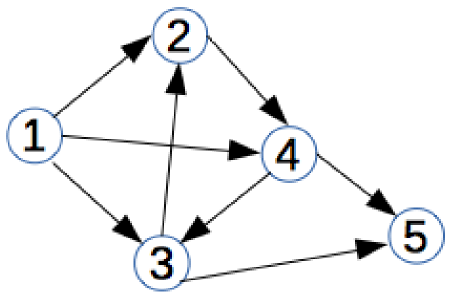
Adjacency Matrix: In graph theory and computer science, an adjacency matrix is a square matrix or a 2-Dimention array used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

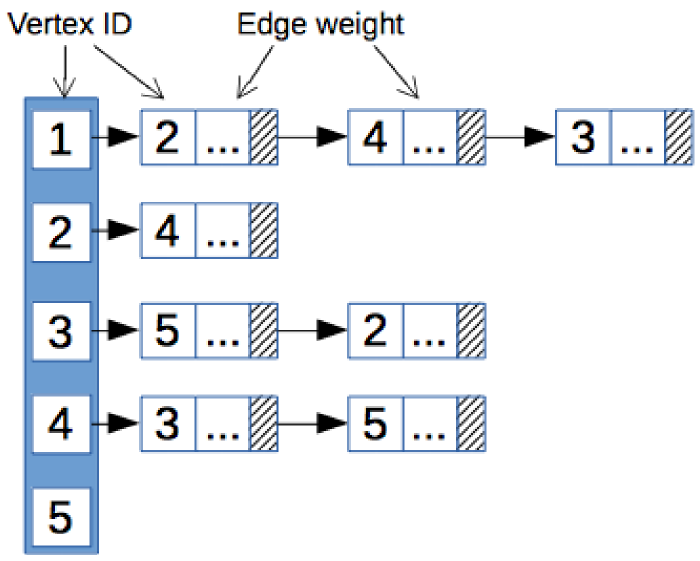


$$\begin{array}{c|ccccc|}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 & 0 \\
3 & 0 & 1 & 1 & 0 & 1 \\
4 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}$$

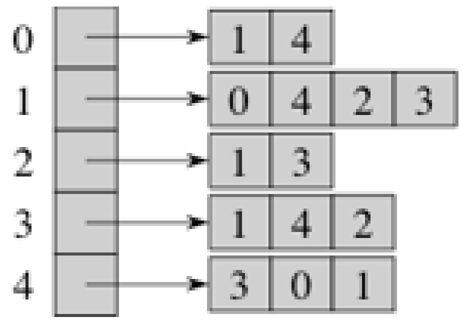
Usually a connection between two nodes is represented with either 0 for “not connected” and 1 for “connected”, but it can also be represented as False or True, 0 or > 1 (which in this case numbers like 2 or 5 or 126 etc. do not only represent the existence of a "connection" but they also show the weight of an edges (connection).

Adjacency List: In graph theory and computer science, an adjacency list is a collection of unordered lists used to represent a finite graph. Each list describes the set of neighbors of a vertex in the graph. This is one of several commonly used representations of graphs for use in computer programs.





The picture above presents an efficient adjacency list meaning that connection references do not repeat themselves (Directed Graph). For example: Node 1 is connected to node 2 and that is being referenced but there is no reference at all when it comes to node to being connected with node 1. That is because the connection is already stated by node 1 and does not repeat itself in node 2 (Undirected Graph). An example of a not so efficient list can be seen bellow.



In this case as we see the connection from node 0 to node 1 is being referenced as a connection from node 1 to node 0 and this situation can lead to several disadvantages.

Adjacency Matrix

Advantages

1) Good for graphs with lot of edges.
2) Faster confirmation of the existence of an edge (connection). ➔ $O(1)$

Disadvantages

1) Inefficient as far as the space is concerned for graphs with number of edges less than the number of nodes. ➔ $O(n^2)$

Adjacency List

Advantages

1) Good for graphs with few edges (edges < nodes), because only the connected nodes will be referenced in the list. ➔ $O(n + m)$
   where n is the number nodes, m is the number of edges.

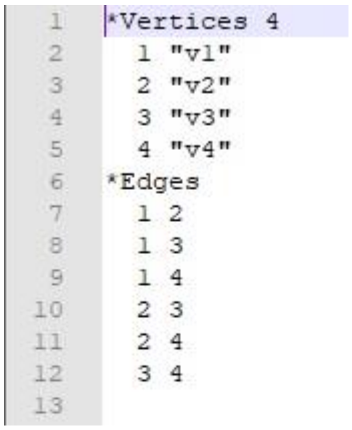
Disadvantages

1) Not so Memory, CPU efficient when it comes to graphs with many nodes and even more edges.
2) Slower confirmation of the existence of an edge (connection). Worst case scenario can take up to ➔ $O(n)$ complexity.

Note: The tool is currently using undirected graphs.

4. Analysis Sector

Moving on with graph analysis the user can analyze the last generated graph either based on adjacency matrix or adjacency list, or he can analyze a pajek file with the following default pajek file format:



```
*Vertices 4
  1 "v1"
  2 "v2"
  3 "v3"
  4 "v4"
*Edges
  1 2
  1 3
  1 4
  2 3
  2 4
  3 4

```

Going back now to the last generated graph option the user can analyze the graph in order to get valuable information about the structure of the graph and he can also come to useful conclusions about different scenarios depending on how the user defines a graph with nodes and edges. For example, there might be a graph and the nodes of the graph can represent humans and the edges of the graph can represent phone calls and therefore can someone realize how much a human (node) is related to another.

Furthermore, there are some other interesting option that the user has and they are the ones that called graph metrics. Geodesic Paths, Closeness Centrality and Betweenness Centrality are available to be applied and calculated on the chosen graph.

4.1 Geodesic Paths

A shortest path between two graph vertices (u, v) of a graph. There may be more than one different shortest paths, all of the same length. Graph geodesics may be found using a **breadth-first traversal** [17] (BFS Moore 1959) or using **Dijkstra's algorithm** [18]. The length of the maximum geodesic in a given graph is called the **graph diameter**, and the length of the minimum geodesic is called the **graph radius**.

4.2 Closeness Centrality

In a connected graph, **closeness centrality** (or **closeness**) of a node is a measure of centrality in a network, calculated as the average of the sum of the length of the shortest paths between the node and all other nodes in the graph. Thus, the more central a node is, the *closer* it is to all other nodes and the smaller his closeness centrality value is.

$$c(n_i) = \sum_{j=1}^{n} \frac{1}{\mathrm{d(ij)}}$$

4.3 Betweenness Centrality

In graph theory, **betweenness centrality** is a measure of centrality in a graph based on shortest paths. For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized. The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex. The higher the betweenness centrality value of a node is means that this node was found in many geodesic paths.

$$b(n_i) = \sum_{j=1}^{n} \frac{D(jl)}{\mathrm{D(ij)}}$$

- Where D_{ij} is the total number of different geodesic paths from node i to node j.

- Where $D_{jl}(i)$ is the total number of geodesic paths that pass-through node i.

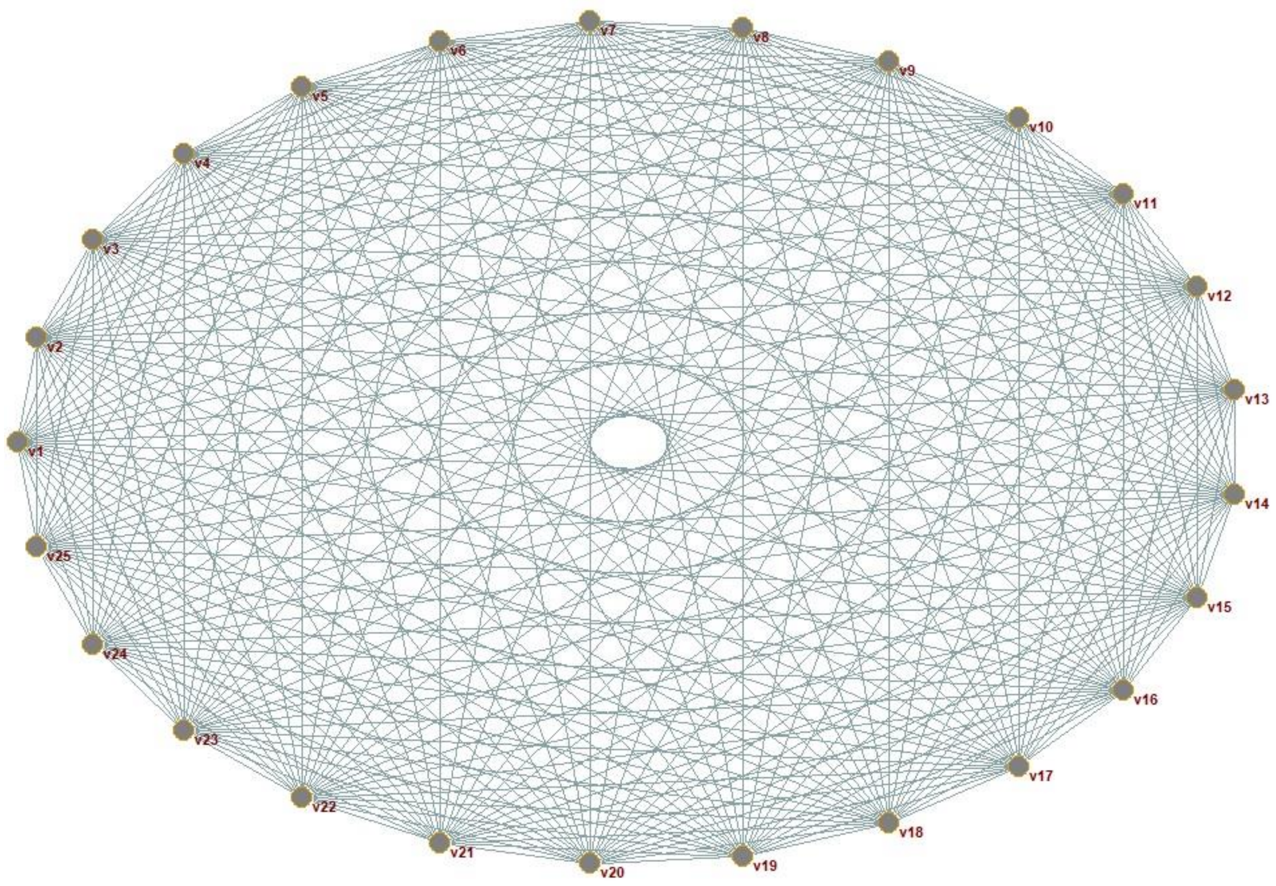
Finally, the user can save the results of the analysis in a txt file with the “save output” button.

5. Visualization Sector

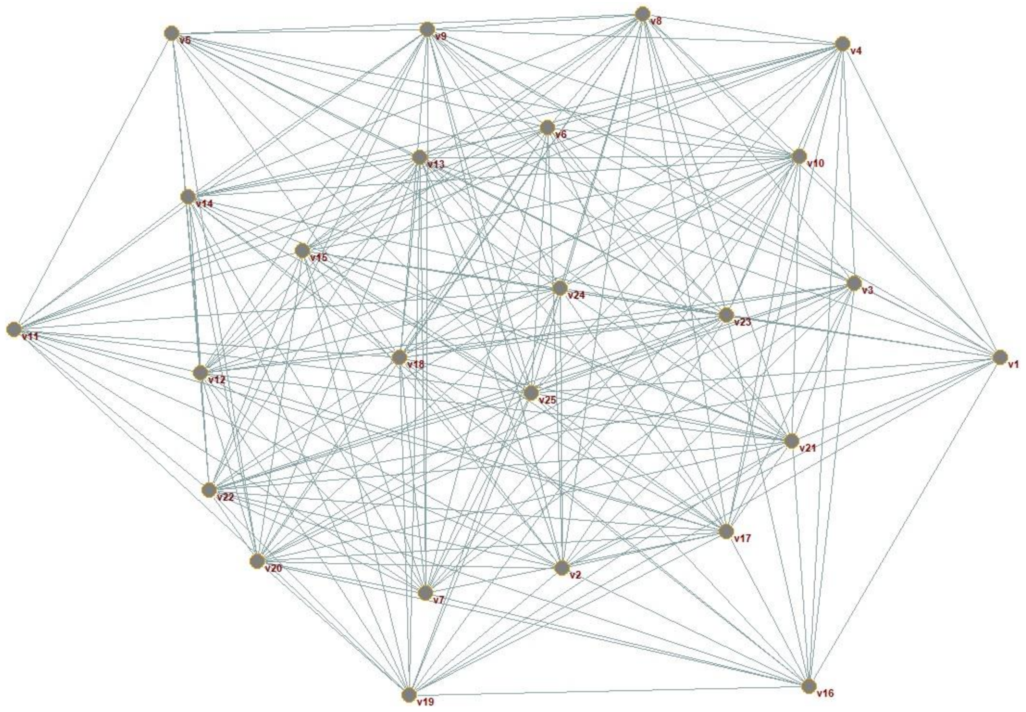
Finally, the user can utilize the visualization sector in order to take a good look on the generated, analyzed graph, have a better understanding of its structure and come to a more useful conclusion.

5.1 Pajek

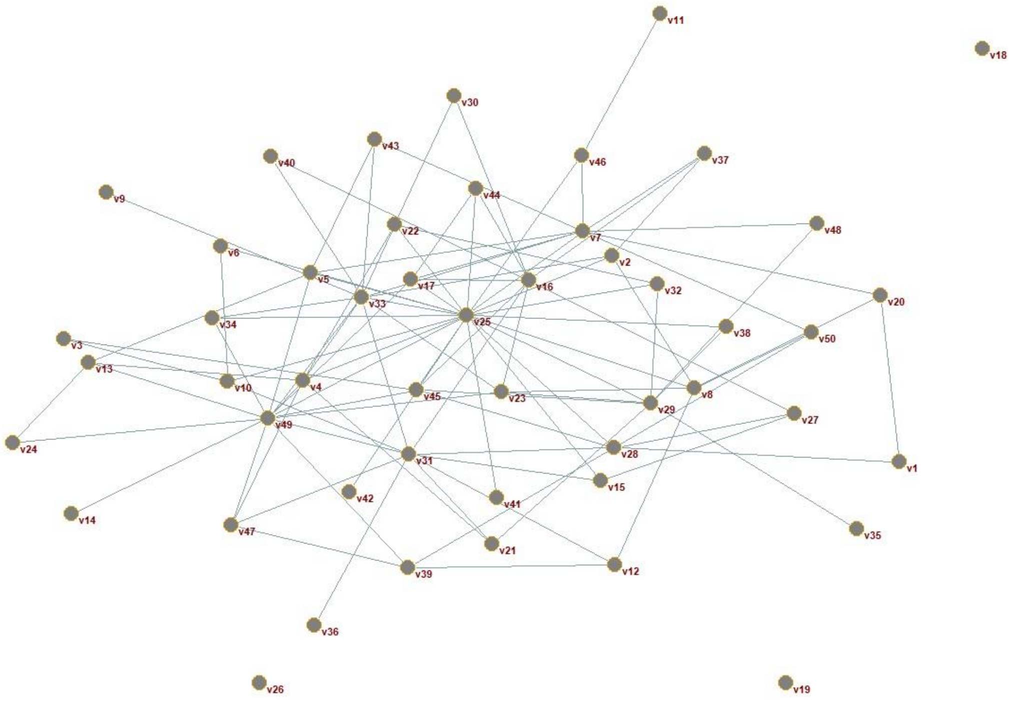
In this section it is provided a choice of visualization in a program called Pajek which is a tool that visualizes graphs in a two-dimension (2D) form. After you visualize the graph in pajek form a .net file is created. You can import this file in Pajek and draw the graph.



25 nodes, Homogeneous Graph.



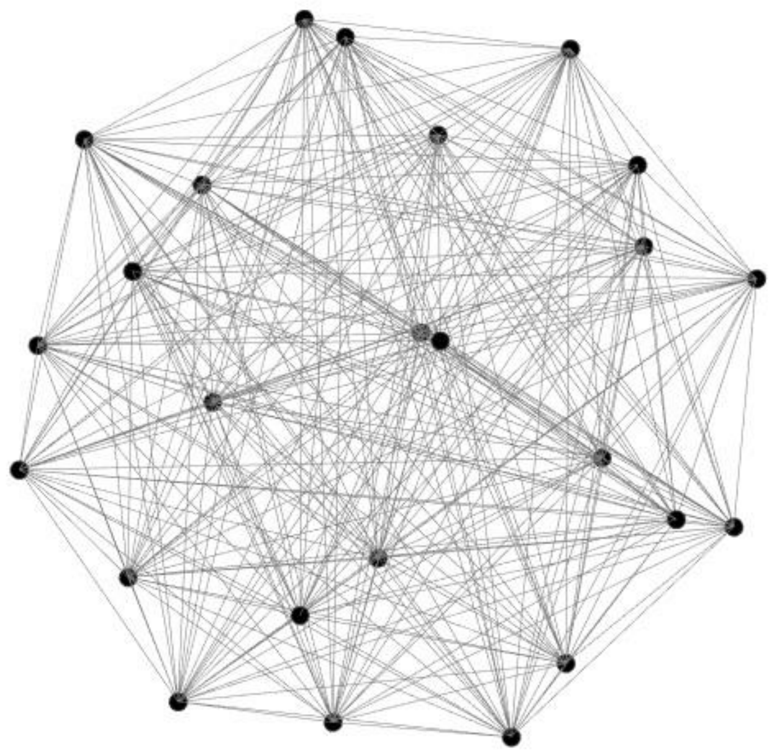
25 nodes, 0.5 probability, 0 seed, ER Graph.



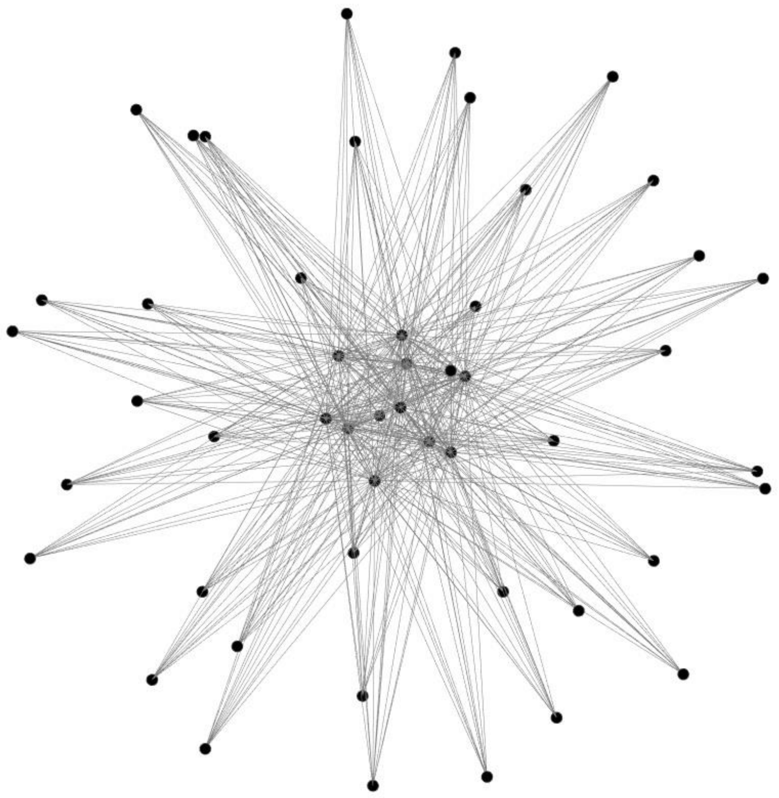
50 nodes, 0 seed, Scale-Free Preferential Attachment only.

5.2 Plotly

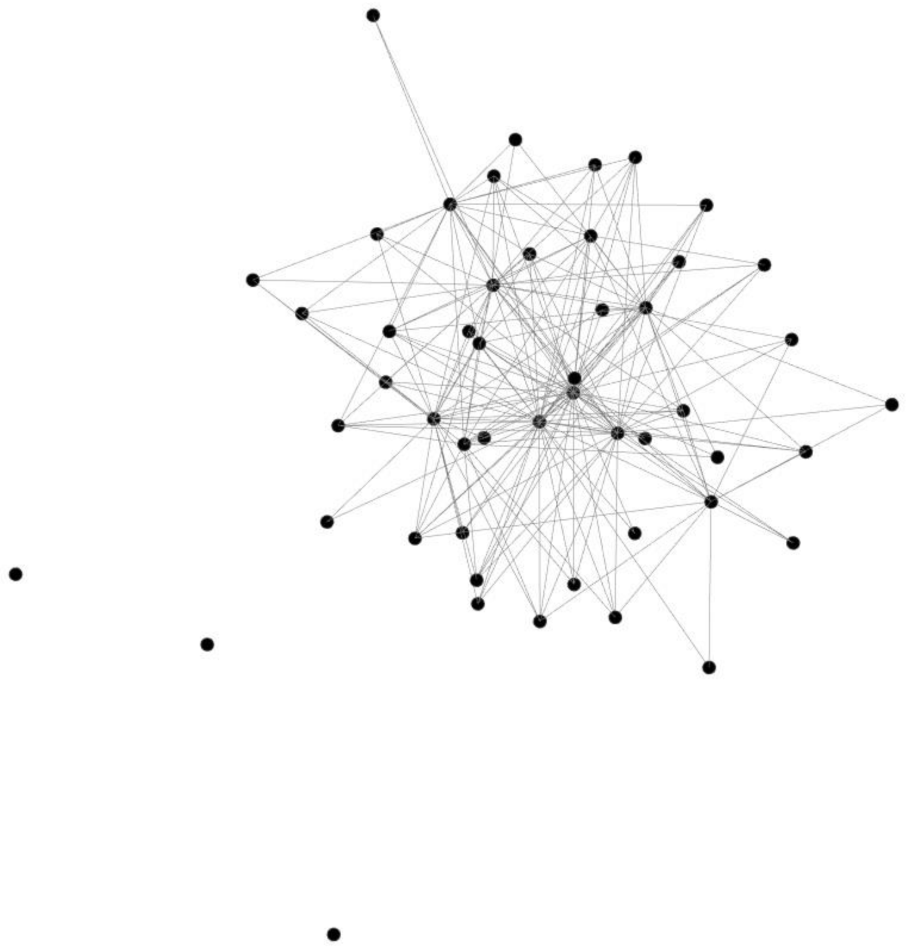
The user can also visualize a graph in a three-dimension (3D) form with **Plotly** which is a technical computing company headquartered in Montreal, Quebec and its website is located here. All the user needs for this type of visualization is to create an account on Plotly and generate an API-KEY from the account that he created.



25 nodes, Homogeneous Graph.



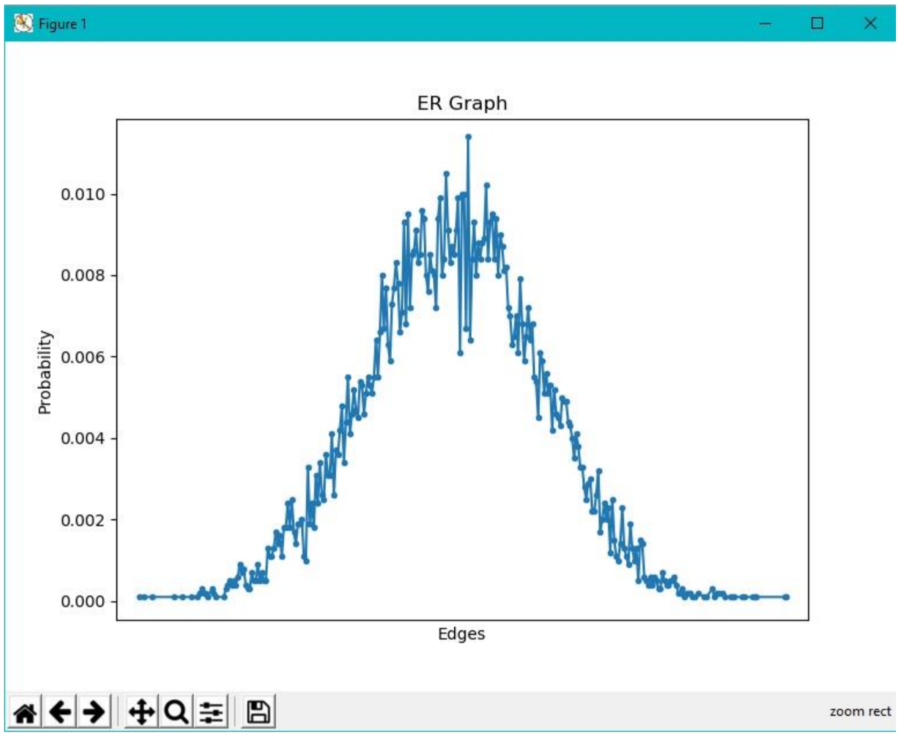
50 nodes, 0.5 probability, 0 seed, ER Graph.



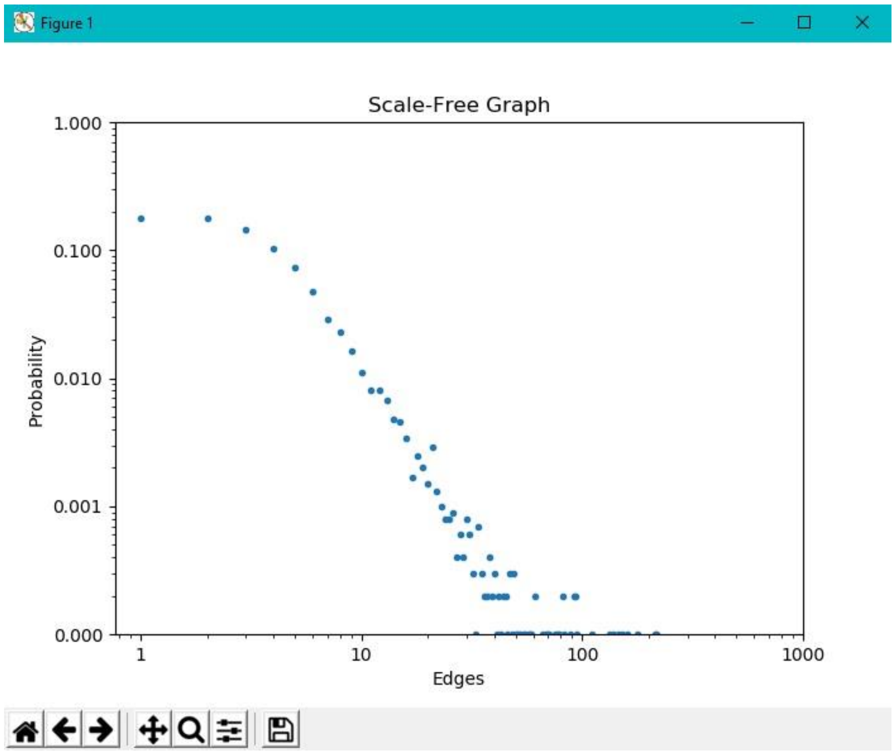
50 nodes, 0 seed, Scale-Free Preferential Attachment only.

5.3 Matplotlib

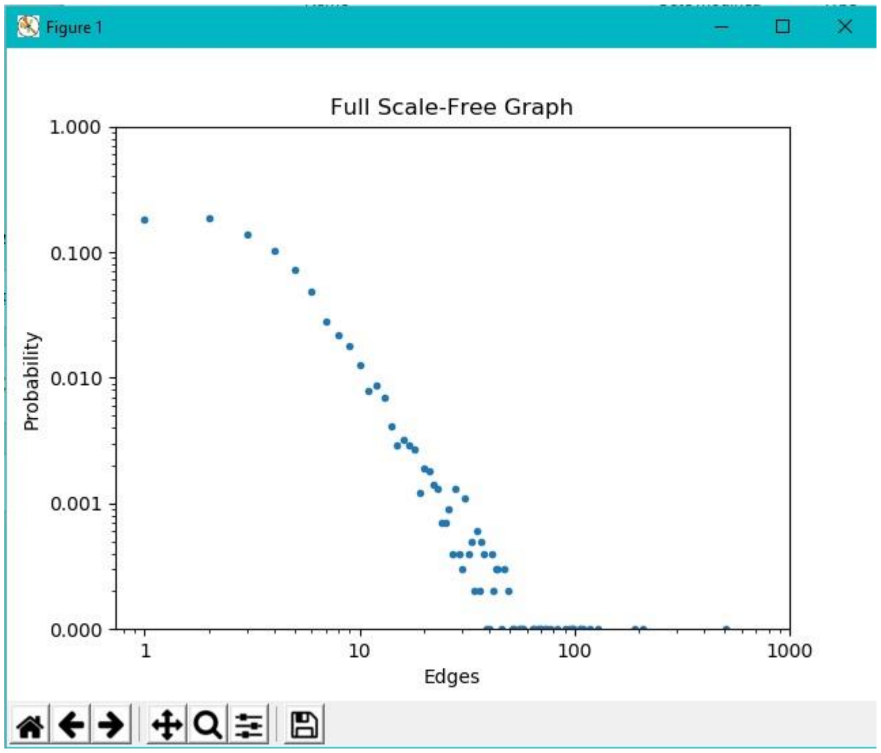
Closing up, the user can also create and observe the graph distribution using the build-in python module named matplotlib.



ER Graph, 10000 nodes, 0.5 probability.



10000 nodes, 0 seed, Preferential attachment only.



10000 nodes, 3 initial nodes, 0 seed, Preferential Attachment and Incremental Growth.

7. Conclusions

The source code of the TOOL_NAME is available on GitHub.

The TOOL_NAME tool has been designed to allow the development of the most utilized network graph topologies in an easy and reproducible way. Our aim is to assist scientists from related fields to create and use graphs without detailed knowledge of graph theory, and give them the possibility to compare their results with previous or current studies without additional effort. We are convinced that with tools, such as the TOOL_NAME, researchers can concentrate their efforts on developing efficient algorithms and understand complex system dynamics, rather than trying to build testing infrastructure.

8. References

https://www.google.com/

https://www.wikipedia.org/

http://ngce.sourceforge.net/