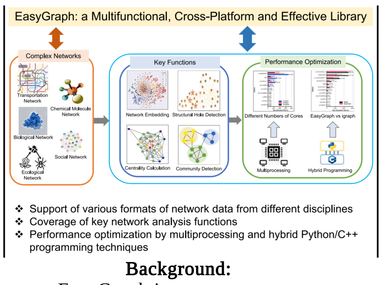
GLOBAL RESEARCH IMMERSION PROGRAM FOR YOUNG SCIENTISTS

EasyGraph: Simplifying Network Analysis with Scalable and Efficient Algorithms

Author: Kun Cheng Supervisor: Yang Chen Affiliation: FuDan Univeristy Home Univ: UC, Santa Barbara

Research Objectives:

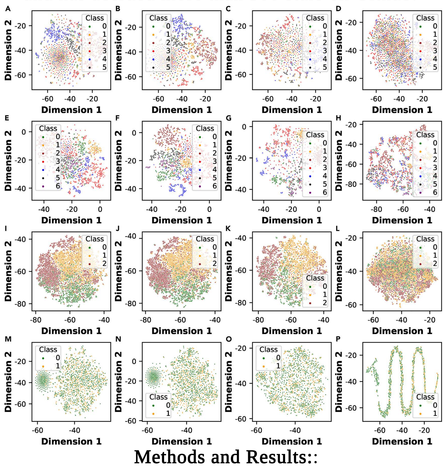
The pipeline of industrial software engineering, graphs and networks, documentation revision and code revision as a test intern on an industrial software product.



Background:

EasyGraph is an open-source network analysis library designed, by FuDan DataNET Group, to cover advanced network processing methods.

It includes functionalities for detecting structural hole spanners, network embedding, and various classic network analysis techniques. As a successful industrial open-source library, EasyGraph has been donwloaded for around 500, 000 times at this point and has accrued more than 300 stars on Github.



Methods and Results::

Why do we need it?

NetworkX is the dominant library in use for network analysis and plotting. While it is an outstanding work in this field, EasyGraph tends to wrap fucntions and datasets together, introduces visualization and a new mechanism in hypergraph analysis, and brings C++ into overall optimization. The graph above showcases the visualization of the results of algorithms for network embedding using t-SNE for different network datasets. This integrated library enables users to deploy and visualize their libraries more conveniently.

EasyGraph extends its powerful network analysis capabilities to include hypergraphs, which provide a more generalized way of representing complex relationships than traditional graphs. In a hypergraph, an edge, known as a hyperedge, can connect any number of nodes, enabling the modeling of multi-party relationships and interactions that are not easily captured by standard pairwise connections. EasyGraph's hypergraph functionality allows users to create, manipulate, and analyze hypergraphs effortlessly. With tools for adding and removing hyperedges, calculating hypergraph-specific metrics, and visualizing hypergraph structures, EasyGraph facilitates the exploration of intricate and higher-order relationships within datasets, making it an invaluable resource for researchers and practitioners working on advanced network analysis problems.

Some sample functions implemented in EasyGraph inlcude: PageRank, Shortest Path Betweenness Centrality, and Closness Centrality.

PageRank is an algorithm originally developed by Google to rank web pages in their search engine results . It assigns a numerical weight to each element of a network (such as web pages), measuring its relative importance. The algorithm is defined by the formula:

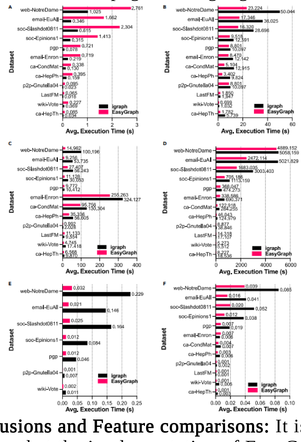
$$PR(u) = \frac{1-d}{N} + d\sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

PageRank operates on the premise that important nodes are likely to be linked by other important nodes. The algorithm can be thought of as a random surfer model,where a surfer randomly clicks on links with probability *d* and jumps to a random node with probability *1- d.*

Discussion:

PageRank is typically computed iteratively until convergence. Starting with an initial PageRank (often uniformly distributed), the algorithm repeatedly updates the PageRank values using the formula above until the values stabilize within a desired threshold.

In the context of networks and graphs, this algorithm analyzes the relationships between nodes and generates a ranking for the nodes, similar to webpage ranking algorithms.



Conclusions and Feature comparisons: It is quite obvious that the implementation of EasyGraph is laconic and efficient, which helps it overpowers other graphing and network Python libraries.

Revision:

classes\:

- classes\graph.py\add_edge -> fixed the problem of passing node_attr as a string parameter
- classes\graph.py\add_node -> fixed the problem of passing node_attr as a string parameter
- classes\graph.py_add_one_node ->
- classes\graph.py\add_edges_from -> prevented tuple objects from being added to a graph
- classes\graph.py_add_one_node -> boundary check on none node
- classes\graph.py\has_node -> boundary check on none node
- classes\graph.py\has_edge -> boundary check on none node
- classes\graph.py\remove_node(self, nodes_to_remove: list) -> boundary check on none node
- classes\directed_multigraph.py\reverse(self, copy=True) -> delete stuff behind self.edges
- classes\directed_multigraph.py\add_edge -> rewrite the passing of parameter *attr*.
- classes\directed_graph.py\add_edges_from -> prevented tuple objects from being added to a graph

datasets\:

- get_sample_graph.py -> import progressbar

functions\:

- ego_betweenness.py\ego_betweenness(G, node) -> Since np.matlib.zeros() has already been deprecated, I replaced it with np.zeros() which is of the same functionality.

Key References:

1. Newman, M.E. (2018). Networks (Oxford University Press). https://doi.org/10.1093/oso/9780198805090.001.0001
2. Newman, M.E., Baraba´ si, A.-L.E., and Watts, D.J. (2006) . The Structure and Dynamics of Networks (Princeton University Press). https://doi.org/10.1515/9781400841356.

Contact Information:

For more information, visit our GitHub repository: https:/github.com/easy-graph

Easygraph Website: https://easy-graph.github.io/

Join our Discord Server: https://discord.gg/ppgcw2wUPg

Thank **Professor Chen** and **PhD. Gao** for instructions throughout the research!