Ogbl-biokg: Biomedical Knowledge Graph

Understanding Dataset

Introduction

Đồ thị: Dataset ogbl-biokg là một Đồ Thị Tri Thức (KG), được tạo ra từ dữ liệu của nhiều kho dữ liệu sinh học lớn.

Nó chứa 5 loại thực thể: diseases, proteins, drugs, sideeffects, functions

Có 51 loại quan hệ có hướng kết nối giữa hai loại thực thể()

- Drugs-protein, drugs-sideeffects, và functions-function được biểu diễn dưới dạng các cạnh có hướng.
- Các quan hệ kết nối cùng loại thực thể (ví dụ, protein-protein, drugs- drugs, functions-functions) biểu diễn bằng các cạnh vô hướng.

Introduction

BioKg đặt ra những thách thức cho các mô hình ML. Cụ thể, nó chứa nhiều thông tin không chính xác (noise) và không đầy đủ (incomplete). Do các Triplet trong KG này được thu thập từ nhiều nguồn khác nhau, mỗi nguồn có mức độ tin cậy khác nhau.

VD: Một số cơ sở dữ liệu của BioKG có dữ liệu được soạn từ các chuyên gia (như SwissProt trong UniProt), trong khi các cơ sở dữ liệu khác có thể chứa dữ liệu được tạo ra từ các kỹ thuật suy diễn. Điều này làm tăng thêm độ phức tạp trong việc xử lý và phân tích dữ liệu.

Triplet: mô tả các mối quan hệ giữa các thực thể sinh học trong mạng đồ thị, chẳng hạn như mối quan hệ giữa 2 protein (TP53, interacts with, MDM2): Protein TP53 tương tác với protein MDM2

Introduction

Các loại quan hệ trong BIOKG:

	EdgeType	Count
0	function-function	1433230
1	protein-function	777577
2	protein-protein_reaction	352546
3	protein-protein_catalysis	303434
4	protein-protein_binding	292254
5	drug-sideeffect	157479
6	drug-protein	117930
7	drug-drug_cardiovascular_system_disease	94842
8	drug-drug_gastrointestinal_system_disease	83210
9	drug-drug_respiratory_system_disease	82168
10	drug-drug_nervous_system_disease	80208
11	drug-drug_hematopoietic_system_disease	79202
12	drug-drug_integumentary_system_disease	73902
13	disease-protein	73547
14	protein-protein_activation	73044
15	drug-drug_urinary_system_disease	67326
16	drug-drug_acquired_metabolic_disease	63430
17	drug-drug_musculoskeletal_system_disease	57926
18	drug-drug_endocrine_system_disease	55994
19	drug-drug_cancer	48514
20	drug-drug_viral_infectious_disease	38846
21	drug-drug_inherited_metabolic_disorder	36492
22	drug-drug_fungal_infectious_disease	36114
23	drug-drug_cognitive_disorder	34660
24	drug-drug_immune_system_disease	34242
25	drug-drug_benign_neoplasm	30348
26	drug-drug_struct_sim	26348
27	drug-drug_sleep_disorder	25860
28	protein-protein_inhibition	25732
29	drug-drug_bacterial_infectious_disease	18554
30	drug-drug_reproductive_system_disease	17006
31	protein-protein_ptmod	15120
32	drug-drug_developmental_disorder_of_mental_health	14314
33	drug-drug_irritable_bowel_syndrome	8528
34	drug-disease	5147
35	drug-drug_thoracic_disease	4660
36	drug-drug_substance-related_disorder	4392
37	drug-drug_pre-malignant_neoplasm	3224
38	drug-drug_hematopoietic_system_diseases	3006
39	drug-drug_somatoform_disorder	2214
40	drug-drug_psoriatic_arthritis	2014
41	protein-protein_expression	1952
42	drug-drug_parasitic_infectious_disease	1680
43	drug-drug_sexual_disorder	1260
44	drug-drug_personality_disorder	972
45	drug-drug_monogenic_disease	600
46	drug-drug_polycystic_ovary_syndrome	514
47	drug-drug_orofacial_cleft	380
48	drug-drug_chromosomal_disease	316
49	drug-drug_hypospadias	292
50	drug-drug_cryptorchidism	128

1. Tổng quan mạng

Nguồn dữ liệu từ Open Graph Benchmark

```
from ogb.linkproppred import LinkPropPredDataset

dataset = LinkPropPredDataset(name = "ogbl-biokg")
graph = dataset[0]
```

```
Downloading http://snap.stanford.edu/ogb/data/linkproppred/biokg.zip
Downloaded 0.90 GB: 100%|██████████| 920/920 [01:02<00:00, 14.82it/s]
```

Tiền sử lý dữ liệu

B1: Convert dataset sang đúng conversion:

```
old_keys = list(edge_index.keys())
for old_name in old_keys:
    new_name = "--".join(old_name)
    edge_index[new_name] = edge_index[old_name]
    del edge_index[old_name]
```

B2: Chuyển sang json object

```
edge_index_json = json.dumps(edge_index, cls = NumpyEncoder)
```

B3: Mapping sang Dataframe các thông tin cần thiết cho đồ thị.

```
# Hàm chuyển đổi danh sách cạnh
def convert_biokg(sub_kg, sub_label):
    split_label = sub_label.split('--')
    origin_label = [f"{split_label[0]}_{x}" for x in sub_kg[0]]
    destination_label = [f"{split_label[2]}_{x}" for x in sub_kg[1]]
    return pd.DataFrame({
        'Origin': origin_label,
        'Destination': destination_label,
        'OriginType': split_label[0],
        'DestinationType': split_label[2],
        'EdgeType': split_label[1]
    })

# Chuyển đổi danh sách cạnh
biokg_edge_list = pd.concat([convert_biokg(biokg[key], key) for key in biokg])

# Tạo danh sách nút
biokg_node_list = pd.DataFrame({
    'Node': pd.concat([biokg_edge_list['Origin'], biokg_edge_list['Destination']]).unique()
})
biokg_node_list['NodeType'] = biokg_node_list['Node'].apply(lambda x: x.split('_')[0])

# Thêm tên nút
biokg_node_list = biokg_node_list.merge(mappings[['Label', 'Name']], left_on='Node', right_on='Label', how='left')
biokg_node_list.rename(columns={'Name': 'NodeName'}, inplace=True)
```

1. Tổng quan mạng

Danh sách nút

```
biokg_node_list
```

	Node	NodeType	Label	NodeName
0	disease_1718	disease	disease_1718	C0038586
1	disease_4903	disease	disease_4903	C0751849
2	disease_5480	disease	disease_5480	C1320474
3	disease_3148	disease	disease_3148	C0270844
4	disease_10300	disease	disease_10300	C4279912
...	...	...	...	...
93768	function_42892	function	function_42892	GO:1905904
93769	function_37594	function	function_37594	GO:0120092
93770	function_37624	function	function_37624	GO:0140068
93771	function_37532	function	function_37532	GO:0106030
93772	function_37082	function	function_37082	GO:0099040

93773 rows × 4 columns

Danh sách cạnh

```
biokg_edge_list
```

	Origin	Destination	OriginType	DestinationType	EdgeType
0	disease_1718	protein_3207	disease	protein	disease-protein
1	disease_4903	protein_13662	disease	protein	disease-protein
2	disease_5480	protein_15999	disease	protein	disease-protein
3	disease_3148	protein_7247	disease	protein	disease-protein
4	disease_10300	protein_16202	disease	protein	disease-protein
...	...	...	...	...	...
352541	protein_13866	protein_11683	protein	protein	protein-protein_reaction
352542	protein_10825	protein_657	protein	protein	protein-protein_reaction
352543	protein_12658	protein_11999	protein	protein	protein-protein_reaction
352544	protein_12028	protein_2482	protein	protein	protein-protein_reaction
352545	protein_2096	protein_2093	protein	protein	protein-protein_reaction

4762678 rows × 5 columns

Số lượng nút

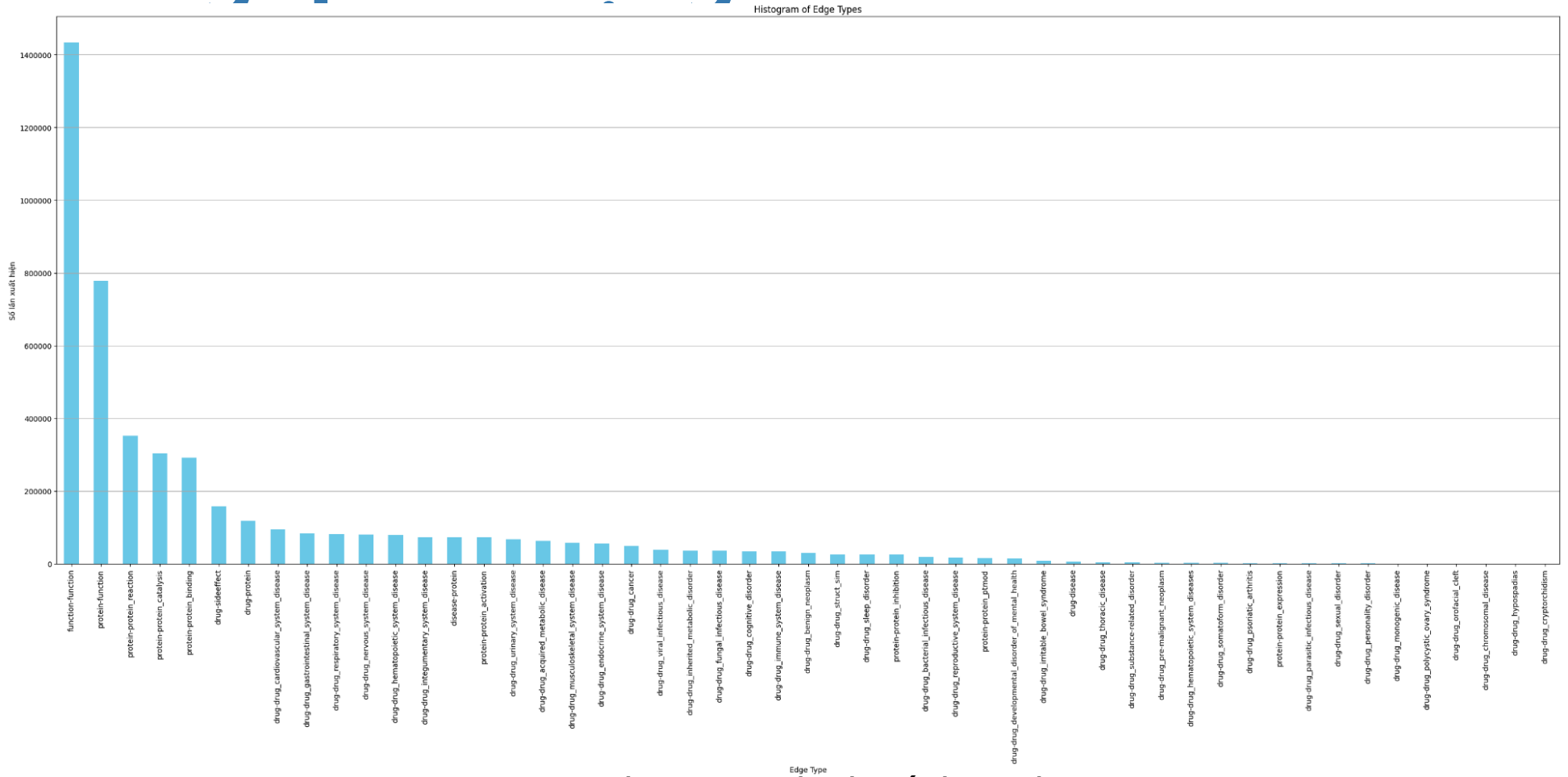
	NodeType	Count
0	disease	10687
1	drug	10533
2	function	45085
3	protein	17499
4	sideeffect	9969

Số lượng cạnh

```
biokg_edge_list.shape[0]
```

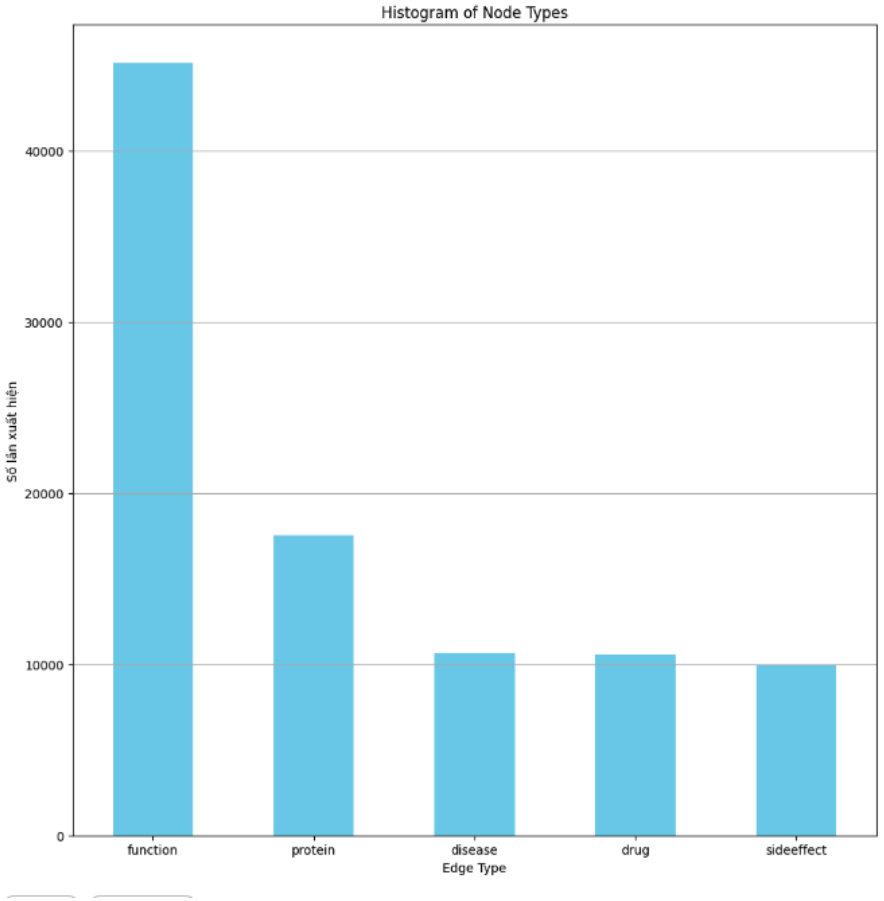
4762678

1. Tổng quan mạng



Histogram danh sách cạnh

1. Tổng quan mạng



Histogram danh sách nút

2. Tổng quan mạng

Visualize ngẫu nhiên 1000 node với k_core = 1,

Với k-core là một đồ thị con trong đó tất cả các đỉnh đều có bậc ít nhất bằng k. Nghĩa là, trong một k-core(đồ thị con), mỗi đỉnh có ít nhất k cạnh kết nối với nó.

2. Cấu trúc mạng

Top 20 Centrality Degree

```
degcent_df = pd.DataFrame(degcent, index=[0]).T
degcent_df.columns = ['degree_centrality']
degcent_df.sort_values('degree_centrality', inplace=True,
ascending=False)
degcent_df.head(20)
```

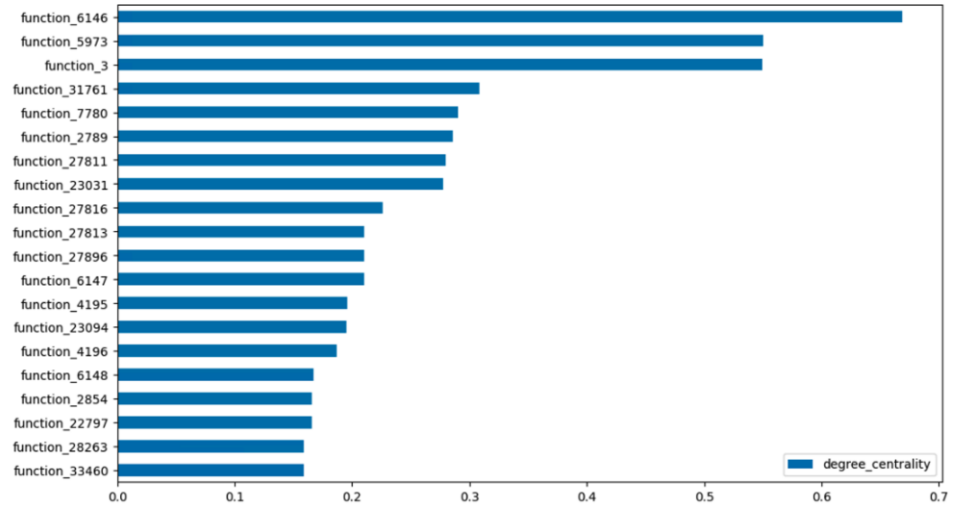
	degree_centrality
function_6146	0.668995
function_5973	0.550484
function_3	0.549759
function_31761	0.308376
function_7780	0.290588
function_2789	0.286184
function_27811	0.279732
function_23031	0.277610
function_27816	0.226155
function_27813	0.210639
function_27896	0.210255
function_6147	0.210041
function_4195	0.195815
function_23094	0.195069
function_4196	0.186527
function_6148	0.167534
function_2854	0.165529
function_22797	0.165508
function_28263	0.159003
function_33460	0.158704

Nhận xét:

Có sự khác biệt đáng kể về độ trung tâm bậc giữa các function. Có node có bậc trung tâm rất cao, trong khi phần lớn các hàm còn lại có độ trung tâm thấp hơn nhiều. Điều này cho thấy mạng lưới có thể có cấu trúc dạng sao hoặc có một số ít "hub" kết nối với nhiều nút khác.

Bậc trung tâm có thể được sử dụng để dự đoán khả năng lan truyền của thông tin hoặc diseases trong mạng.

```
import matplotlib.pyplot as plt
_= degcent_df[0:20].plot.barh(figsize=(12,7))
plt.gca().invert_yaxis()
```



2. Cấu trúc mạng

Top 20 Centrality Degree

```
pagerank_df = pd.DataFrame(pagerank, index=[0]).T
pagerank_df.columns = ['pagerank']
pagerank_df.sort_values('pagerank', inplace=True,
ascending=False)
pagerank_df.head(20)
```

	pagerank
function_6146	0.012236
function_5973	0.011269
function_3	0.011249
function_2789	0.010485
function_4195	0.009338
function_2854	0.006849
function_4196	0.006449
function_6330	0.005397
function_22797	0.005278
function_31761	0.004925
function_23031	0.004865
function_7780	0.004414
function_27811	0.004340
function_27813	0.003967
function_22758	0.003920
function_27896	0.003860
function_6147	0.003847
function_27816	0.003315
function_28263	0.002947
function_23094	0.002924

Nhận xét:

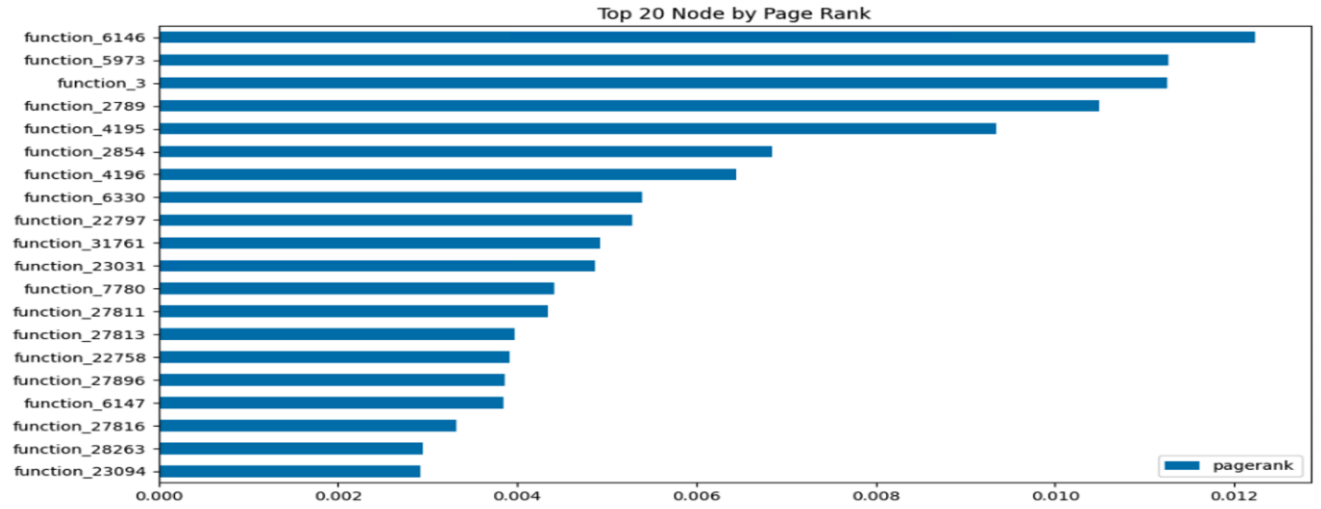
1.Ảnh hưởng của chức năng: function_6146, function_5973, và function_3 có giá trị PageRank cao nhất cho thấy chúng đóng vai trò trung tâm trong mạng lưới này và đại diện cho các chức năng quan trọng được liên kết với các chức năng khác.

2.Phân bố ảnh hưởng: Mức độ ảnh hưởng của các function không đồng đều, phần lớn các function có ảnh hưởng thấp cho thấy cấu trúc mạng lưới có dạng hình sao, với một số trung tâm và nhiều nút ngoại vi.

3.Ứng dụng trong nghiên cứu thuốc:

1. Xác định mục tiêu thuốc
2. Dự đoán tác dụng phụ
3. Tìm kiếm thuốc mới có cơ chế tác động tương tự như các loại thuốc hiện có

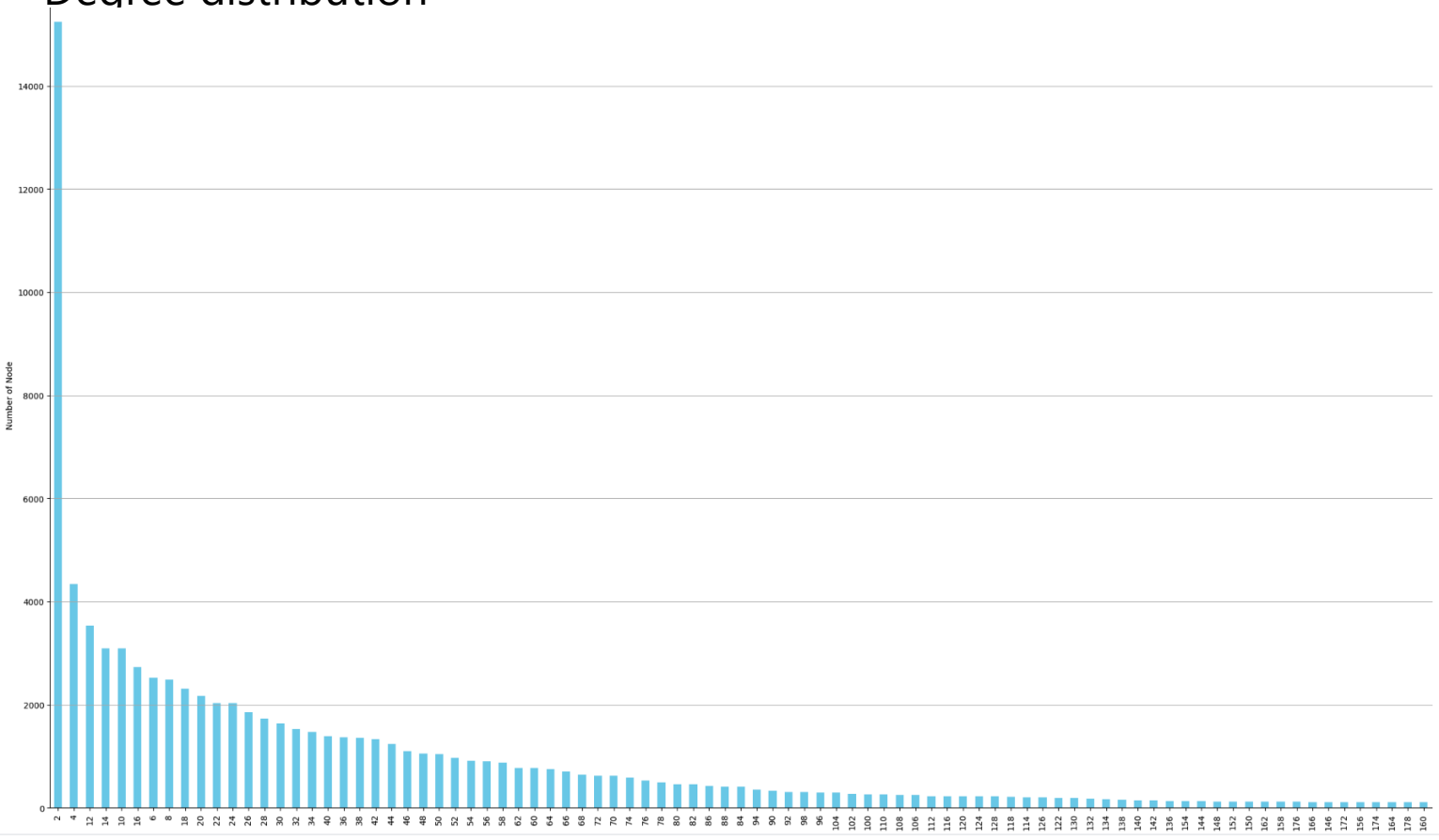
```
title = 'Top 20 Node by Page Rank'
_= pagerank_df[0:20].plot.barh(title=title,
figsize=(12,7))
plt.gca().invert_yaxis()
```



+ Code + Markdown

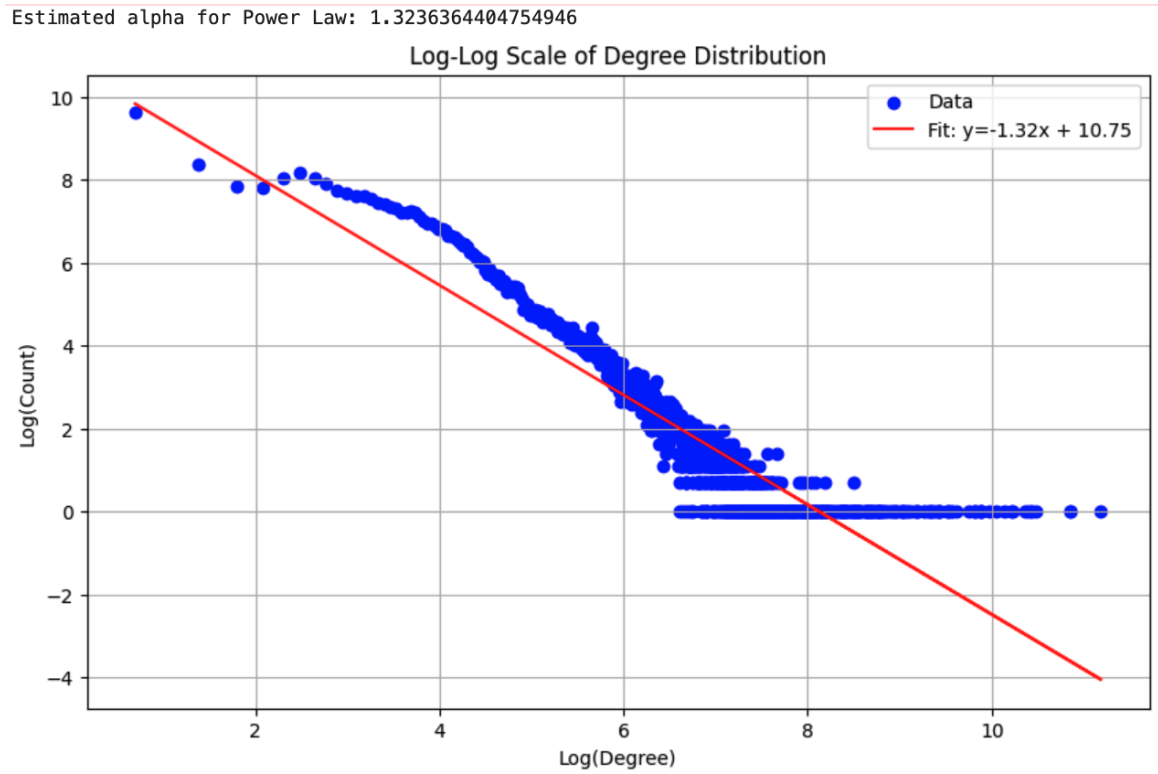
2. Cấu trúc mạng

Degree distribution



2. Cấu trúc mạng

Power law degree distribution



α=1.32 cho thấy rằng số lượng đỉnh với bậc cao giảm nhanh chóng.

Giá trị α nhỏ hơn 2 báo hiệu rằng phân phối bậc có thể dẫn đến một số lượng nhỏ đỉnh (hubs) có nhiều kết nối, trong khi phần lớn đỉnh có ít kết nối hơn. Đồng thời với một số ít đỉnh nhiều kết nối có vai trò quan trọng trong việc kết nối các thực thể khác của mạng.

Đường hồi quy có độ dốc âm, cho thấy khi bậc tăng lên, số lượng đỉnh có bậc đó giảm xuống.

3. Phương pháp xác định cộng đồng

Using the Louvain method

Using label propagation

Using the Girvan-Newman algorithm.

The Louvain method

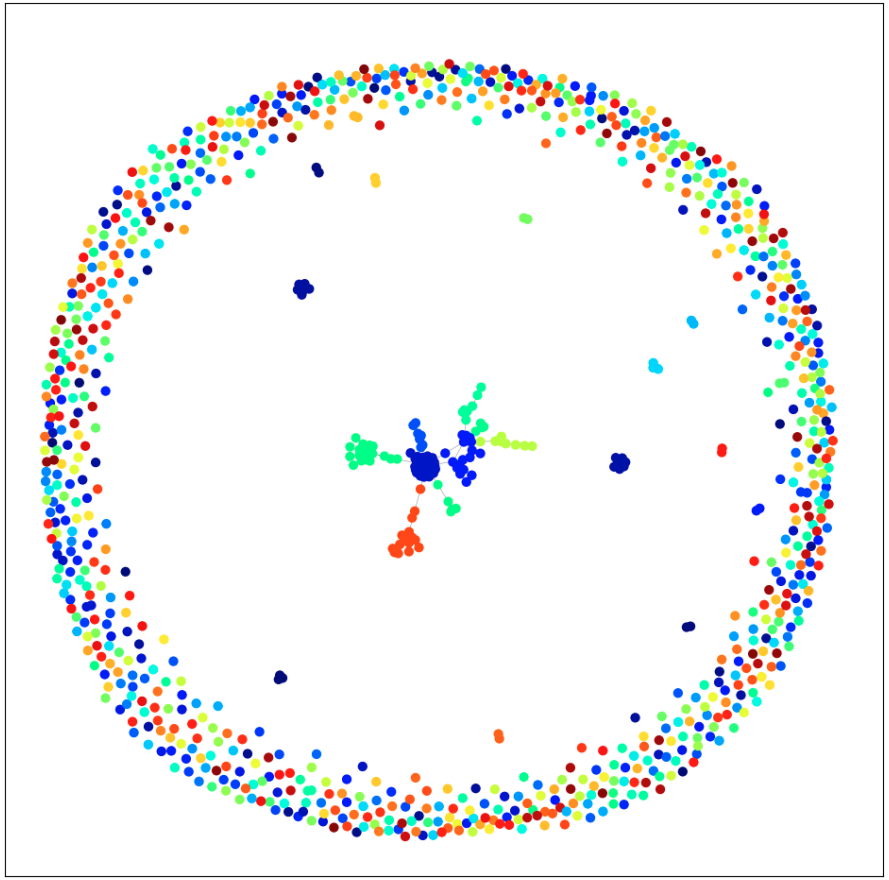
Thuật toán Louvain có thể xử lý hàng trăm triệu nút và hơn một tỷ cạnh

- Sử dung với đồ thị vô hướng
- Sử dụng hàm best_partition để xác định phân vùng tối ưu.
- Hàm draw_patition sẽ tô màu các nút theo cộng đồng mà chúng thuộc về. Mỗi nút thuộc về một phân vùng khác nhau và các phân vùng đó là các công đồng

[1] https://arxiv.org/pdf/0803.0476.pdf

```
# Chuyển đổi sang đồ thị vô hướng
G_undirected = biokg_sample.to_undirected()
partition = community_louvain.best_partition(G_undirected, resolution=1)
draw_partition(G_undirected, partition)
```

```
/tmp/ipykernel_30/667368384.py:8: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotli
maps.get_cmap(obj)`` instead.
  cmap = cm.get_cmap('jet', max(partition.values()) +1)
```



The label propagation

Label propagation là một cách tiếp cận nhanh khác để xác định các cộng đồng tồn tại trong một mạng lưới.

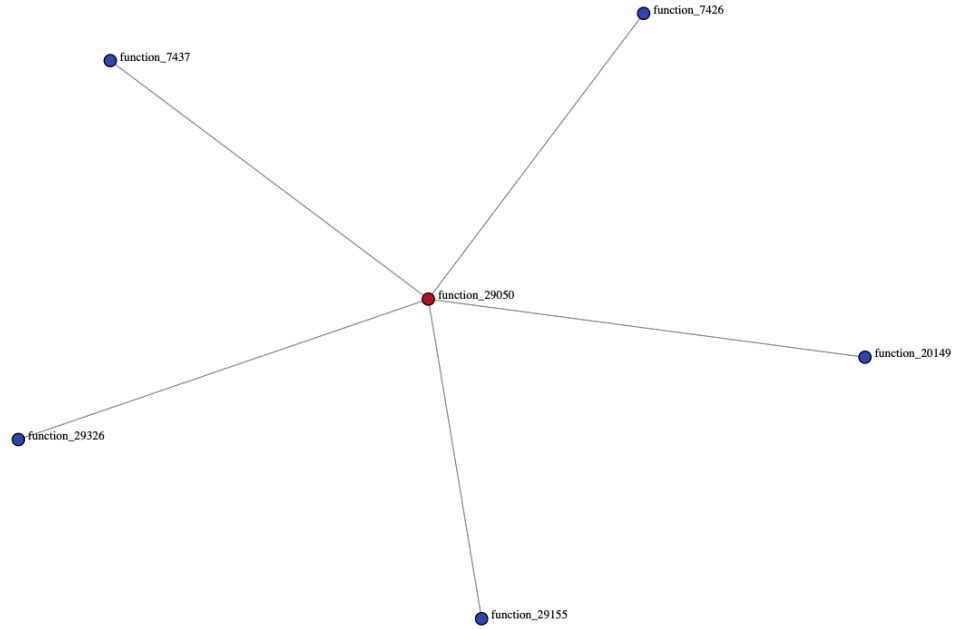
- Phương pháp này cho kết quả không tốt bằng phương pháp Louvain
- Sử dụng **label_propagation_communities** để chuyển đồ thị thành danh sách nhãn từ đó xác định cộng đồng theo nhãn.
- Lọc bỏ đi các node không có cạnh nào để tạo ma trận kề

[2] https://arxiv.org/pdf/0709.2938.pdf

```
from networkx.algorithms.community.label_propagation import label_propagation_communities
communities = label_propagation_communities(G_undirected)
communities = list(communities)
comm_df = pd.DataFrame(communities)
filtered_df = comm_df[comm_df.iloc[:, 1:].notna().sum(axis=1) > 2]
filtered_df
```

	0	1	2	3	4	5	6	7	8	9	...	17	18
17	protein_13226	protein_11341	protein_14833	protein_12477	protein_11835	None	None	None	None	None	...	None	None
35	function_29326	function_29155	function_7437	function_20149	function_29050	function_7426	None	None	None	None	...	None	None
38	function_22546	function_38778	function_4994	function_16767	function_38659	function_4755	function_29457	None	None	None	...	None	None
60	protein_5829	protein_11272	protein_8772	protein_9095	protein_4266	protein_8414	protein_3265	protein_13455	protein_12516	protein_9353	...	protein_7984	protein_16238
70	function_33076	function_39324	protein_2946	protein_6224	function_33202	protein_1488	protein_14299	None	None	None	...	None	None
98	function_7424	function_26522	protein_1170	protein_9606	function_7687	None	None	None	None	None	...	None	None

```
community = communities[35]
G_community = G_undirected.subgraph(community)
draw_graph(G_community, show_names=True, node_size=5)
```



The Girvan-Newman algorithm.

Thuật toán Girvan-Newman xác định các cộng đồng bằng cách cắt cạnh liên kết 2 nút kết nối 2 cộng đồng, dẫn đến việc chia mạng thành hai phần.

- Nhược điểm việc cắt cạnh liên kết sẽ làm mất thông tin về các mối quan hệ, nhưng tất cả các nút vẫn giữ nguyên.
- Sử dụng hàm girvan_newman để chia cộng đồng. Tuy nhiên phải lựa chọn cộng đồng tối ưu nhất. Sử dụng dataframe để lọc

[3] https://www.pnas.org/doi/full/10.1073/pnas.122653799.

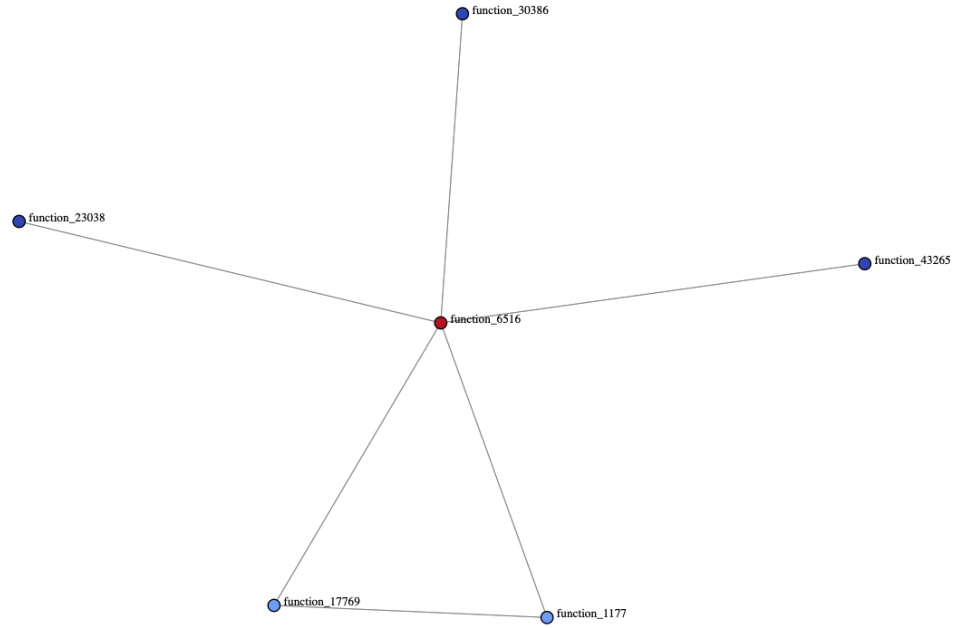
```
from networkx.algorithms.community import girvan_newman
```

```
G_undirected1 = biokg_sub.to_undirected()
communities1 = girvan_newman(G_undirected1)
communities1= list(communities1)
```

```
comm_df = pd.DataFrame(communities1)
filtered_df = comm_df[comm_df.iloc[:, 1:].notna().sum(axis=1) > 2]
filtered_df
```

	0	1	2	3	4	5	6	7	8
2	{function_6079}	{function_5763}	{function_6480}	{function_1177, function_43265, function_6516,...	None	None	None	None	None
3	{function_6079}	{function_5763}	{function_6480}	{function_1177, function_6516, function_17769,...	{function_43265}	None	None	None	None
4	{function_6079}	{function_5763}	{function_6480}	{function_17769, function_30386, function_2303...	{function_43265}	{function_6516}	None	None	None

```
community1 = communities1[2]
community1= community1[3]
G_community1 = G_undirected.subgraph(community1)
draw_graph(G_community1, show_names=True, node_size=5)
```



Nhận xét

Phương pháp Louvain chắc chắn là phương pháp tối ưu nhất

- Thuật toán này có thể được sử dụng trên các mạng rất lớn gồm hàng triệu nút và nó sẽ hiệu quả và nhanh
- Có thể tinh chỉnh tham số độ phân giải để tìm ra phân vùng tốt nhất cho cộng đồng phát hiện, mang lại sự linh hoạt khi kết quả mặc định không tối ưu. Với các thuật toán khác không có sự linh hoạt này.
- Louvain cho thấy tổng qua đa phần nhiều node chưa được liên kết với nhau

Questions & answers