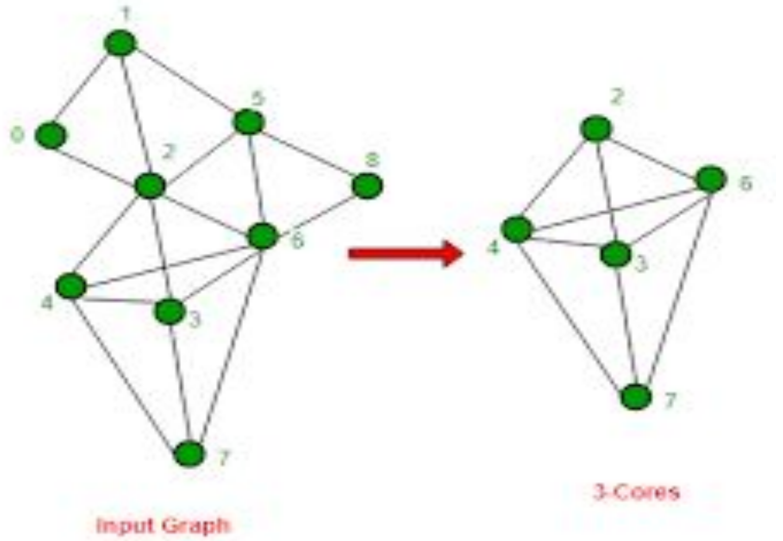
Maximum K-Core Subgraph of a Graph

Ritik Pansuriya(181IT237)
Karthik R(181IT235)
Shidharth S(181IT243)
Yash Parakh(181IT253)

Problem Statement

To find the highest value of K-core of graph, i.e., a subgraph of a graph where each node is of at least degree K



Introduction

K-Core analysis is widely adopted to find the densest part of a network across a broad range of scientific subjects. Such studies have been applied in a variety of settings, including real networks like the Internet topology, social networks like co-authorships graphs, protein networks in bioinformatics, and so on.

Larger values of “coreness”, correspond to nodes with a more central position in the network structure. Given that cores with larger k are known to be good spreaders, this information could be used at run-time to optimize the diffusion of information in the network.

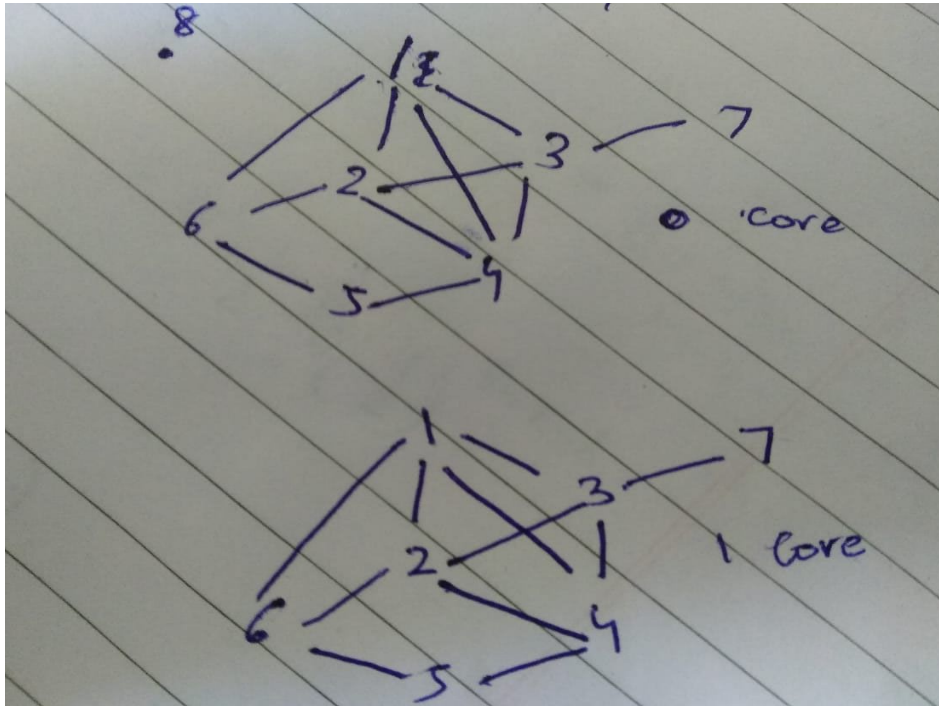
Literature Survey

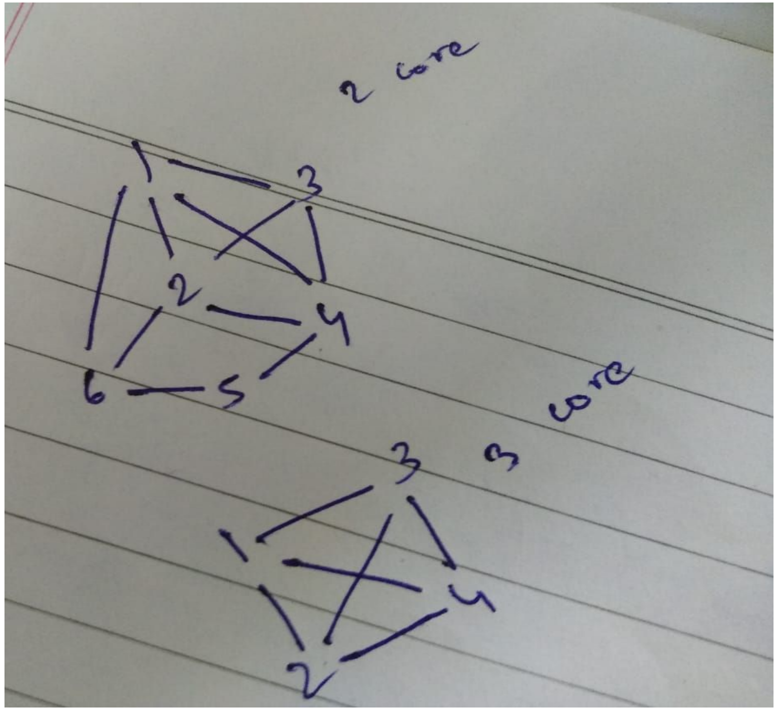
- An O(m) Algorithm for Cores Decomposition of Network, by Vladimir Batagelj and Matjaž Zaveršnik

- Smallest Last Ordering & Clustering and Graph Coloring Algorithms, by David Matula and Leland Beck

Methodology

The k-core of a graph G is the maximal subgraph where the degree of the vertex H >= k. K-core of a graph is unique for a given value k.



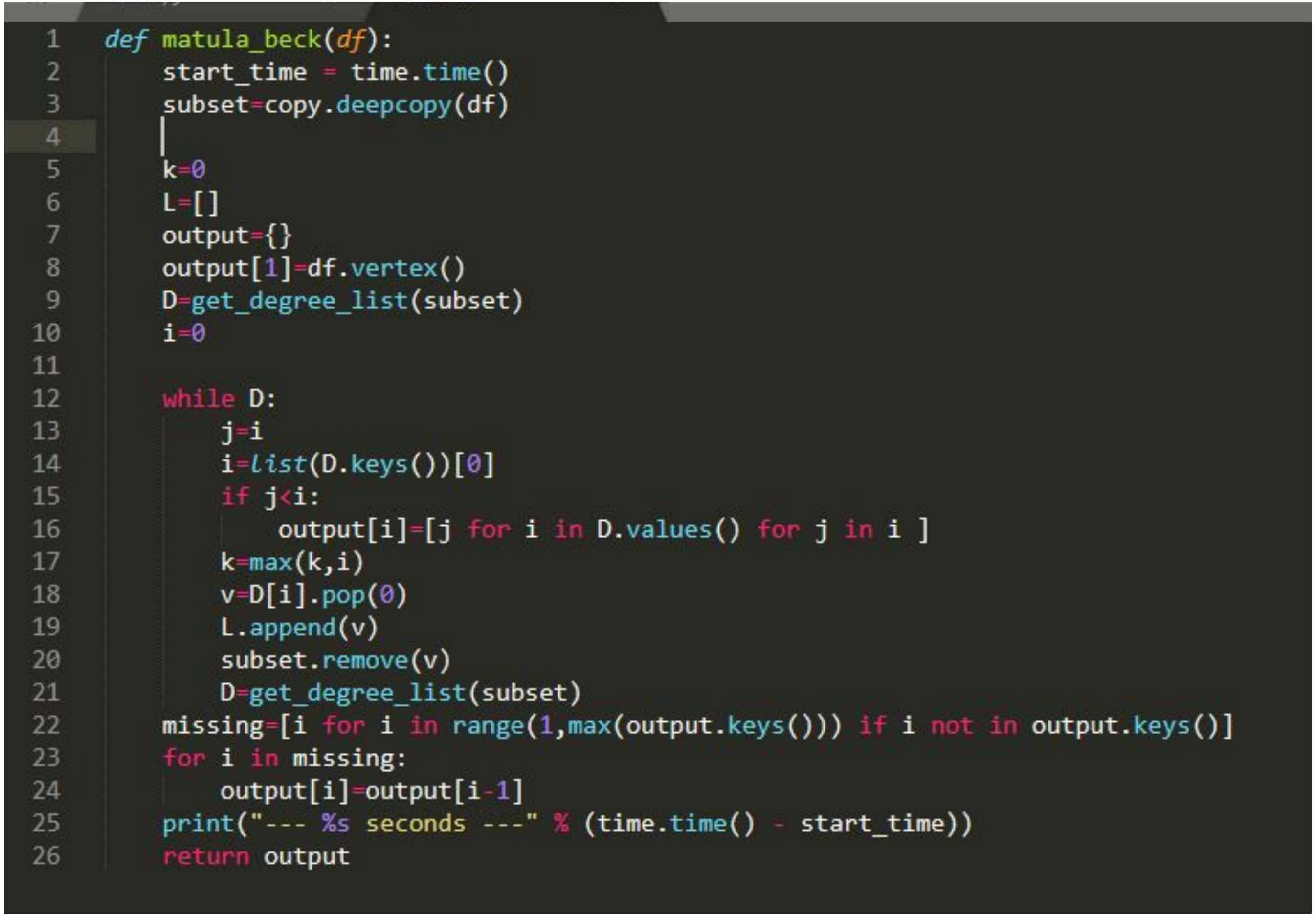


David Matula & Leland Beck Approach

- Initialize an output list *L*.
- Compute a number d_v for each vertex *v* in *G*, the number of neighbors of *v* that are not already in *L*. Initially, these numbers are just the degrees of the vertices.
- Initialize an array *D* such that *D*[*i*] contains a list of the vertices *v* that are not already in *L* for which $d_v = i$.
- Initialize *k* to 0.
- Repeat *n* times:
    - Scan the array cells *D*[0], *D*[1], ... until finding an *i* for which *D*[*i*] is nonempty.
    - Set *k* to max(*k*,*i*)
    - Select a vertex *v* from *D*[*i*]. Add *v* to beginning of *L* and remove it from *D*[*i*].
    - For each neighbor *w* of *v* not already in *L*, subtract one from d_w and move *w* to the cell of D corresponding to the new value of d_w.

Code Explanation through Snippets

Code snippet of **Matula - Beck** Approach



```
def matula_beck(df):
    start_time = time.time()
    subset=copy.deepcopy(df)

    k=0
    L=[]
    output={}
    output[1]=df.vertex()
    D=get_degree_list(subset)
    i=0

    while D:
        j=i
        i=list(D.keys())[0]
        if j<i:
            output[i]=[j for i in D.values() for j in i ]
        k=max(k,i)
        v=D[i].pop(0)
        L.append(v)
        subset.remove(v)
        D=get_degree_list(subset)
    missing=[i for i in range(1,max(output.keys())) if i not in output.keys()]
    for i in missing:
        output[i]=output[i-1]
    print("--- %s seconds ---" % (time.time() - start_time))
    return output
```

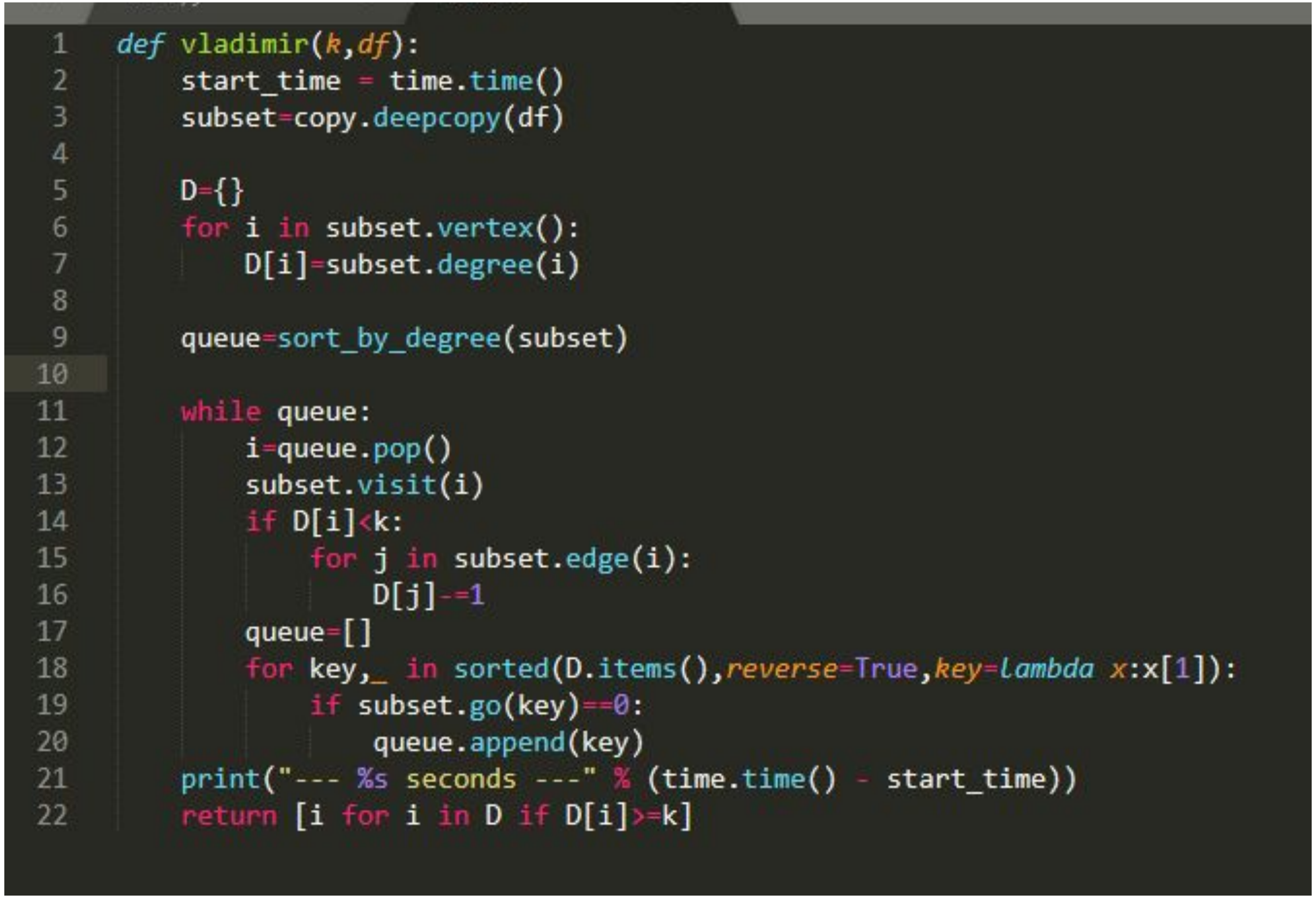
Vladimir Batagelj & Matjaž Zaveršnik Approach

- Compute the degrees of vertices
- Order the set of vertices V in increasing order of their degrees
- For each v ∈ V in the order do begin

```
core[v] := degree[v]
for each u ∈ Neighbors(v) do
    if degree[u] > degree[v] then begin
        degree[u] := degree[u] − 1
        reorder V accordingly
```

Code Explanation through Snippets

Code snippet of **Batagelj - Zaveršnik** Approach



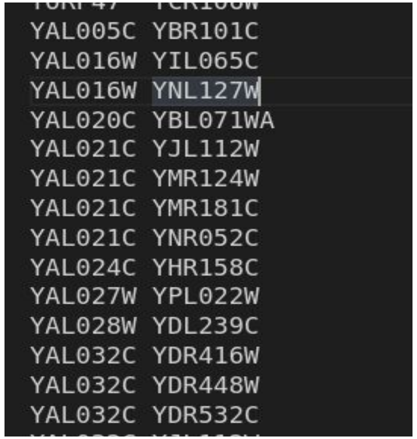
```
def vladimir(k,df):
    start_time = time.time()
    subset=copy.deepcopy(df)

    D={}
    for i in subset.vertex():
        D[i]=subset.degree(i)

    queue=sort_by_degree(subset)

    while queue:
        i=queue.pop()
        subset.visit(i)
        if D[i]<k:
            for j in subset.edge(i):
                D[j]-=1
        queue=[]
        for key,_ in sorted(D.items(),reverse=True,key=lambda x:x[1]):
            if subset.go(key)==0:
                queue.append(key)
    print("--- %s seconds ---" % (time.time() - start_time))
    return [i for i in D if D[i]>=k]
```

Time Complexity

David Matula & Leland Beck Approach: $O(|E|+|V|)$

Vladimir Batagelj & Matjaž Zaveršnik Approach : $O(\max(|E|,|V|))$

Main.py



```
YAL005C YBR101C
YAL016W YIL065C
YAL016W YNL127W
YAL020C YBL071WA
YAL021C YJL112W
YAL021C YMR124W
YAL021C YMR181C
YAL021C YNR052C
YAL024C YHR158C
YAL027W YPL022W
YAL028W YDL239C
YAL032C YDR416W
YAL032C YDR448W
YAL032C YDR532C
```

Functions in the main.py file:

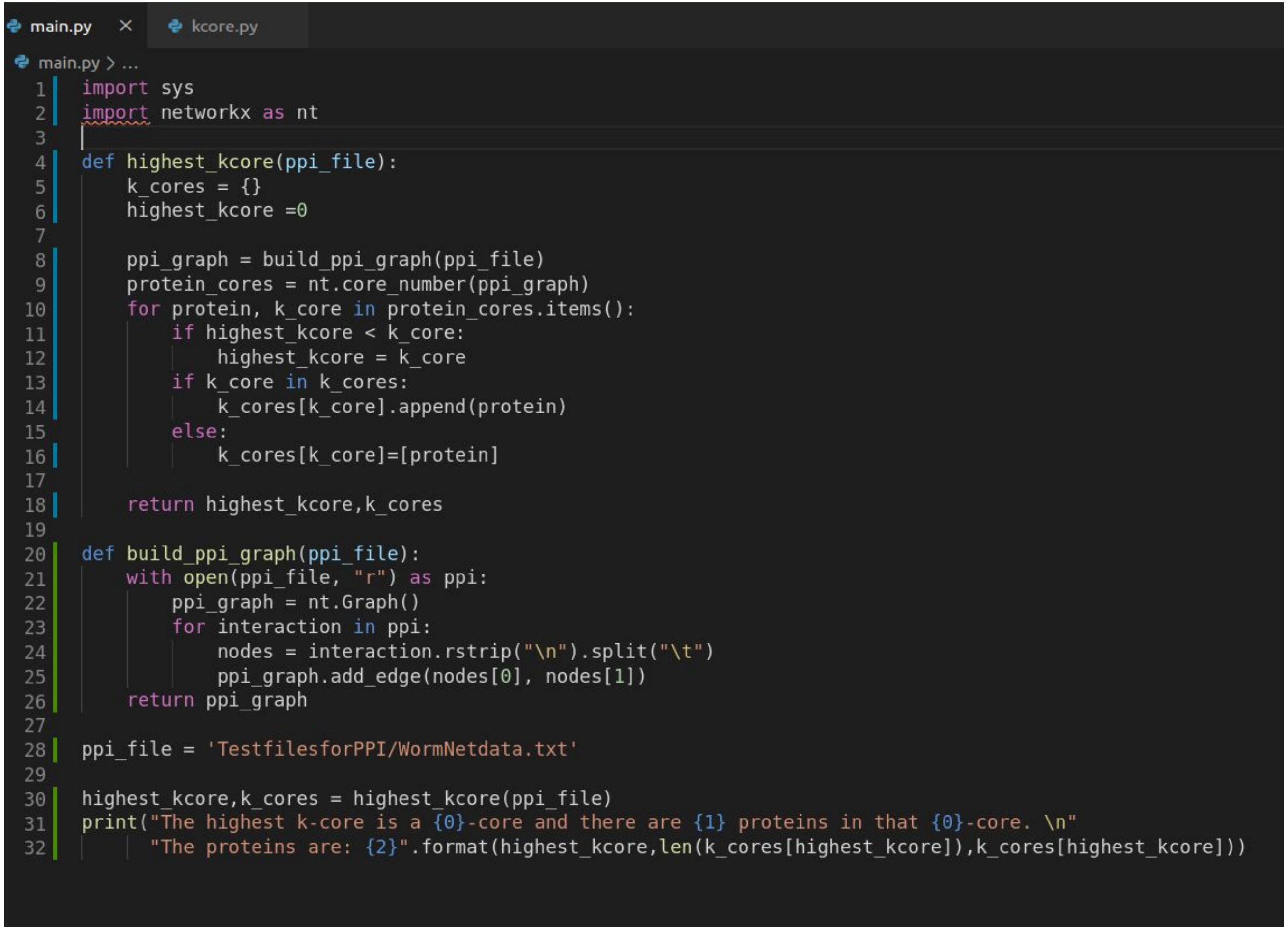
1. Build_network
2. Find_kcores

- build_ppi_graph: Builds the protein-protein interaction graph, strips any extra space in the input file and splits the nodes. Imports the Graph class from networkx library for building the network.

- core_number: Computes the max k-core subgraph possible from the network for each protein such that it is included in the subgraph.

Code Explanation through Snippets

Code snippet of **main.py file**



main.py × kcore.py

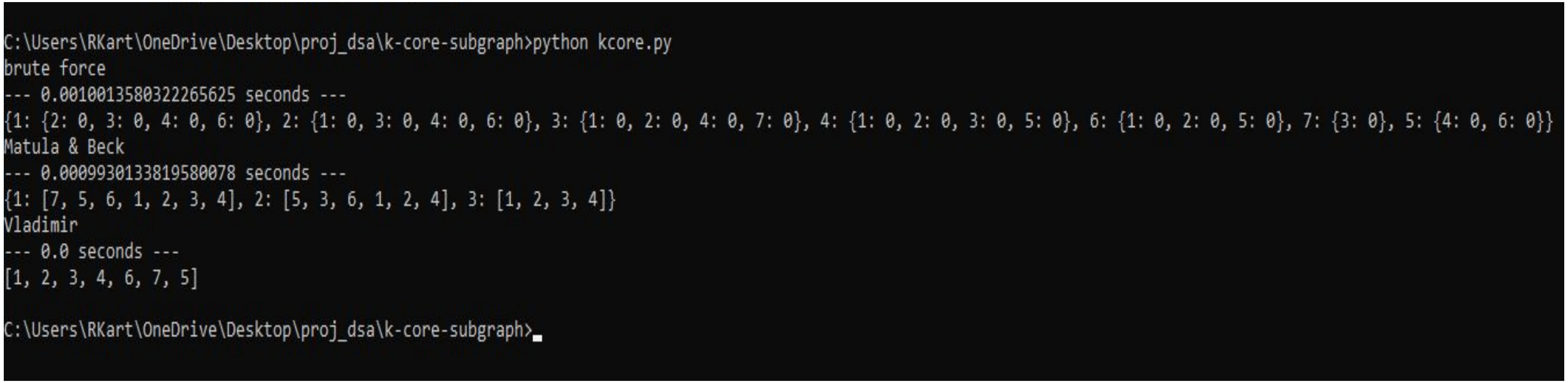
main.py > ...

```python
import sys
import networkx as nt

def highest_kcore(ppi_file):
    k_cores = {}
    highest_kcore =0

    ppi_graph = build_ppi_graph(ppi_file)
    protein_cores = nt.core_number(ppi_graph)
    for protein, k_core in protein_cores.items():
        if highest_kcore < k_core:
            highest_kcore = k_core
        if k_core in k_cores:
            k_cores[k_core].append(protein)
        else:
            k_cores[k_core]=[protein]

    return highest_kcore,k_cores

def build_ppi_graph(ppi_file):
    with open(ppi_file, "r") as ppi:
        ppi_graph = nt.Graph()
        for interaction in ppi:
            nodes = interaction.rstrip("\n").split("\t")
            ppi_graph.add_edge(nodes[0], nodes[1])
    return ppi_graph

ppi_file = 'TestfilesforPPI/WormNetdata.txt'

highest_kcore,k_cores = highest_kcore(ppi_file)
print("The highest k-core is a {0}-core and there are {1} proteins in that {0}-core. \n"
      "The proteins are: {2}".format(highest_kcore,len(k_cores[highest_kcore]),k_cores[highest_kcore]))
```

Results

Comparison between

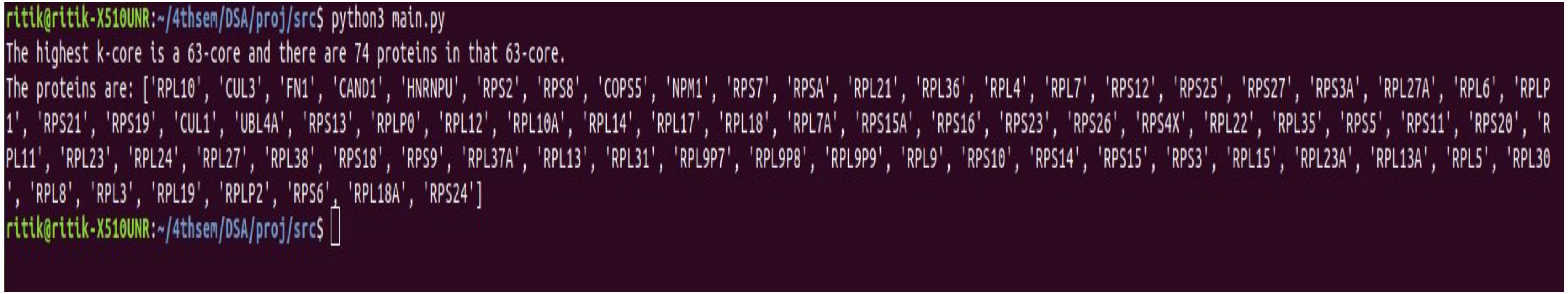
- Brute Force
- David Matula & Leland Beck Approach
- Vladimir Batagelj & Matjaž Zaveršnik Approach



```
C:\Users\RKart\OneDrive\Desktop\proj_dsa\k-core-subgraph>python kcore.py
brute force
--- 0.0010013580322265625 seconds ---
{1: {2: 0, 3: 0, 4: 0, 6: 0}, 2: {1: 0, 3: 0, 4: 0, 6: 0}, 3: {1: 0, 2: 0, 4: 0, 7: 0}, 4: {1: 0, 2: 0, 3: 0, 5: 0}, 6: {1: 0, 2: 0, 5: 0}, 7: {3: 0}, 5: {4: 0, 6: 0}}
Matula & Beck
--- 0.0009930133819580078 seconds ---
{1: [7, 5, 6, 1, 2, 3, 4], 2: [5, 3, 6, 1, 2, 4], 3: [1, 2, 3, 4]}
Vladimir
--- 0.0 seconds ---
[1, 2, 3, 4, 6, 7, 5]

C:\Users\RKart\OneDrive\Desktop\proj_dsa\k-core-subgraph>_
```

Results

- Output of our main.py file.



```
ritik@ritik-X510UNR:~/4thsem/DSA/proj/src$ python3 main.py
The highest k-core is a 63-core and there are 74 proteins in that 63-core.
The proteins are: ['RPL10', 'CUL3', 'FN1', 'CAND1', 'HNRNPU', 'RPS2', 'RPS8', 'COPS5', 'NPM1', 'RPS7', 'RPSA', 'RPL21', 'RPL36', 'RPL4', 'RPL7', 'RPS12', 'RPS25', 'RPS27', 'RPS3A', 'RPL27A', 'RPL6', 'RPLP
1', 'RPS21', 'RPS19', 'CUL1', 'UBL4A', 'RPS13', 'RPLP0', 'RPL12', 'RPL10A', 'RPL14', 'RPL17', 'RPL18', 'RPL7A', 'RPS15A', 'RPS16', 'RPS23', 'RPS26', 'RPS4X', 'RPL22', 'RPL35', 'RPS5', 'RPS11', 'RPS20', 'R
PL11', 'RPL23', 'RPL24', 'RPL27', 'RPL38', 'RPS18', 'RPS9', 'RPL37A', 'RPL13', 'RPL31', 'RPL9P7', 'RPL9P8', 'RPL9P9', 'RPL9', 'RPS10', 'RPS14', 'RPS15', 'RPS3', 'RPL15', 'RPL23A', 'RPL13A', 'RPL5', 'RPL30
', 'RPL8', 'RPL3', 'RPL19', 'RPLP2', 'RPS6', 'RPL18A', 'RPS24']
ritik@ritik-X510UNR:~/4thsem/DSA/proj/src$ 
```

Applications

- Network Modeling & Analysis

Concept of core decomposition can has been used to study the resilience or robustness of a network.

- Detection of Influential Spreaders

Detecting influential spreaders is important for understanding how information diffuses in social networks, individuals with high connectivity would contribute more in the diffusion process.

- Neuroscience

Cores are used in-depth analysis of the brain functional network, composed of parts of the brain that are functionally interconnected in a dense manner.

References

- The input files we used for the protein protein interaction network are present here : https://www.inetbio.org/wormnet/downloadnetwork.php
- https://dl.acm.org/doi/pdf/10.1145/2402.322385 - David Matula and Leland Beck Approach
- https://arxiv.org/pdf/cs/0310049.pdf - Vladimir Batagelj and Matjaž Zaveršnik Approach